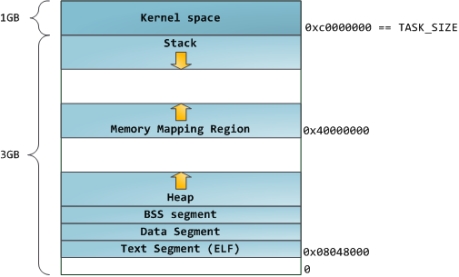
之前在培训gdb调试时有介绍到glibc中关于内存分配的内容，但是不够详细，所以就拿来《glibc内存管理ptmalloc源码分析.pdf》来拜读，收获颇多，遂将学习到的东西在此记录下来。

全书以作者在项目中遇到的内存暴增为出发点来介绍了相关的基础知识和ptmalloc内存管理以及ptmalloc源码的分析。其中的源码是前面基础知识的代码体现。

下面就将书中的重要内容进行摘要并消化。

一、基础知识

Linux系统再装载elf文件时会调用loader把可执行文件的各个段依次载入到某一地址空间中，已32位处理器为例，.text端、.data段、.bss段依次被载入（如下图所示）。



由上图可知应用程序可以访问的地址至0xbfffffff，也即3G，3G-4G空间由内核使用，应用程序不可直接访问。应用程序的栈由高地址向低地址生长，堆是由低地址向高地址生长。Heap和mmap需要映射才可以使用，应用程序可以直接调用系统调用来使用，Stack则不需要映射。

Linux内核在用户申请内存时，只是分配一个虚拟内存，只有当用户使用这块内存时才会将宝贵的物理内存提供给用户。虚拟内存和物理内存存在一定的映射关系(Linux驱动相关书籍有详细介绍)。

可以使用malloc()和free()来申请或释放heap。Mmap的申请和释放则需要mmap()和munmap()。

一、内存管理

a)  基础知识

当不知道程序的每个部分将来需要多少内存时，系统内存空间有限，而内存需求是动态的，这就需要内存管理程序来负责内存的分配和回收。动态性越强，内存管理就越重要。

         常见的内存管理方法如下：

1、 C风格内存管理程序

2、  内存池

3、  引用计数

4、  垃圾回收

以上内存管理方法各有优缺点，不同的内存管理方法都有最擅长的情形。

目前最常用的还是C内存管理程序。常见的C内存管理程序有Doug Lea Malloc、BSD Malloc、Hoard、TCMalloc。其中ptmalloc2和TCMalloc是其中的佼佼者。尤其是TCMalloc，该内存管理系统由Google开发，与标准的Glibc中的malloc相比，TCMalloc在内存分配效率和速度有绝对优势，因为TCMalloc直接与OS打交道。

b)  ptmalloc内存管理

ptmalloc实现了malloc()和free()以及一组其他函数，已提供动态内存管理。分配器处在用户态和内核态之间，它响应用户的分配情况，向操作系统申请内存，然后返回给用户程序，为了保持高效分配，分配器一般都会预先分配一块大于用户请求的内存，并通过某种算法来管理这块内存。分配器不但要管理自己分配的内存还要管理空闲的内存，当响应用户分配的要求时，分配器会首先在空闲空间中寻找一块合适的内存给用户，在空闲空间找不到的情况下才分配一块新的内存。

Ptmalloc在设计时折中了效率、空间利用率、可用性，所以假设了一些情形：

1、  具有长生命周期的大内存分配使用mmap.

2、  特别大的内存分配使用mmap

3、  具有短生命周期的内存分配使用brk.

4、  尽量只缓存临时使用的空闲小内存块，对大内存块或是长生命周期的大内存块在释放时直接归还给操作系统。

5、  对空闲的小内存块只会在malloc和free的时候进行合并，free时空闲内存块可能放入pool中，不一定归还给操作系统。

6、  需要保存长期存储的程序不适合使用ptmalloc

在了解了ptmalloc设计时假设的情形有利于指导我们在实际使用中正确的选用申请内存的函数，并对内存释放有了一定了解。

内存分配有主分配区和非主分配区。

主分配区和非主分配区形成一个环形链表进行管理。

主分配区：

可以访问进程的heap和mmap映射区域，也即主分配区可以使用sbrk()和mmap()向操作系统申请虚拟内存。如果主分配区的内存是通过mmap()向系统申请的，那么当free()，主分配区会直接调用munmap()将内存还给操作系统。

非主分配区：

只能访问进程的mmap映射区，非主分配区每次使用mmap()向系统申请1MB的虚拟内存，当用户向非主分配区申请是再分成小块内存。

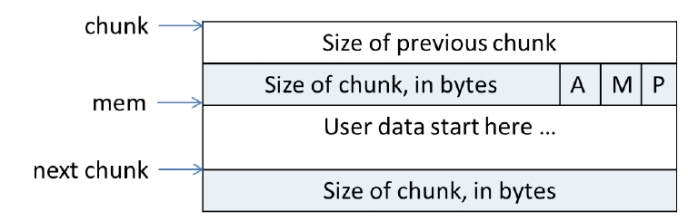
每个进程只有一个主分配区，但是可能存在多个非主分配区，ptmalloc根据系统对分配区的征用情况动态增加非主分配区的数量，分配区数量一旦增加，就不会再减少。

Chunk的组织

不管内存是在哪里被分配，用什么方法分配，用户请求分配的空间在ptmalloc中都是用chunk来表示。用户调用free()函数释放掉内存并不是立即就归还给操作系统，它也被表示为一个chunk，ptmalloc使用特定的数据结构来管理这些空闲的chunk。

Chunk格式

Ptmalloc在给用户非配的空间的前后加上了一些控制信息，用这样的方法来记录分配的信息，以便完成分配和释放工作。一个使用中的chunk（使用中，没有被free）在内存中如下图所示：

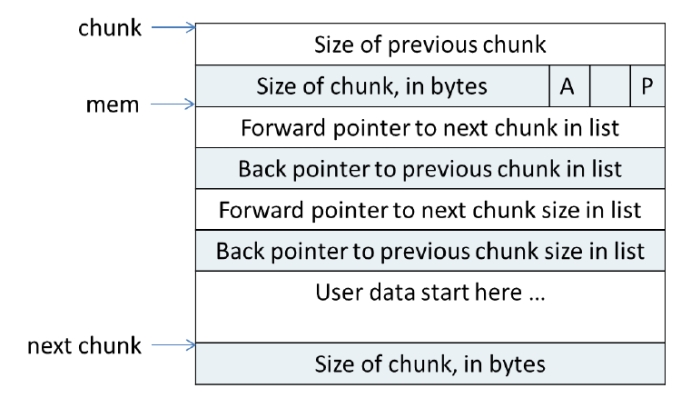


在上图中，chunk指针指向一个chunk的开始，一个chunk中包含了用户请求的内存区域和相关的控制信息。图中的mem指针才是真正返回给用户的内存指针。chunk的第二个域的最低一位为P，它表示前一个块是否在使用中，P为0则表示前一个chunk为空闲，这时 chunk的第一个域prev\_size才有效，prev\_size表示前一个chunk的size，程序可以使用这个值来找到前一个chunk的开始地址。当P为1时，表示前一个chunk正在使用中，prev\_size无效，程序也就不可以得到前一个chunk的大小。不能对前一个chunk进行任何操作。ptmalloc分配的第一个块总是将P设为1，以防止程序引用到不存在的区域。

Chunk的第二个域的倒数第二个位为M，他表示当前chunk是从哪个内存区域获得的虚拟内存。M为1表示该chunk是从mmap映射区域分配的，否则是从heap区域分配的。

Chunk的第二个域倒数第三个位为A，表示该chunk属于主分配区或者非主分配区，如果属于非主分配区，将该位置为1，否则置为0。

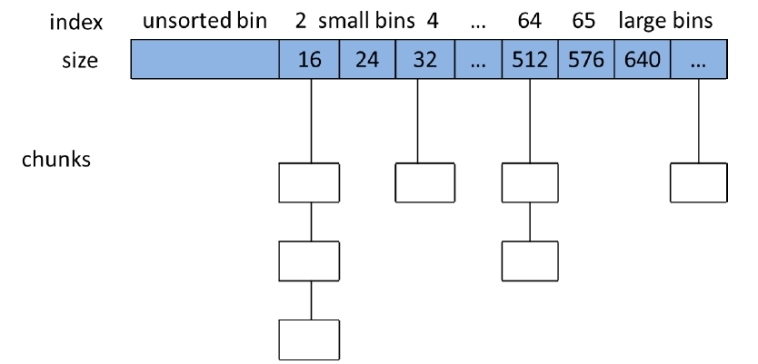
空闲的chunk和使用中的chunk略有不同。空闲的chunk在内存中的结构如下图所示：



当chunk空闲时，其M状态不存在，只有AP状态，原本是用户数据区的地方存储了四个指针，指针fd指向后一个空闲的chunk，而bk指向前一个空闲的chunk，ptmalloc通过这两个指针将大小相近的chunk连成一个双向链表。对于large bin中的空闲chunk，还有两个指针，fd\_nextsize和bk\_nextsize，这两个指针用于加快在large bin中查找最近匹配的空闲chunk。不同的chunk链表又是通过bins或者fastbins来组织的。

在空闲的chunk介绍中提到了bins，bins是用来管理空闲chunk的。Bins主要分为unsorted bin、small bins、large bins。

用户free掉的内存并不是都会马上归还给系统，ptmalloc会统一管理heap和mmap映射区域中的空闲的chunk，当用户进行下一次分配请求时，ptmalloc会首先试图在空闲的chunk中挑选一块给用户，这样就避免了频繁的系统调用，降低了内存分配的开销。ptmalloc将相似大小的chunk用双向链表链接起来，这样的一个链表被称为一个bin。Ptmalloc一共维护了128个bin，并使用一个数组来存储这些bin（如下图所示）。



数组中的第一个为unsorted bin，数组中从2开始编号的前64个bin称为small bins，同一个small bin中的chunk具有相同的大小。两个相邻的small bin中的chunk大小相差8bytes。small bins中的chunk按照最近使用顺序进行排列，最后释放的chunk被链接到链表的头部，而申请chunk是从链表尾部开始，这样，每一个chunk 都有相同的机会被ptmalloc选中。Small bins后面的bin被称作large bins。large bins中的每一个bin分别包含了一个给定范围内的chunk，其中的chunk按大小序排列。相同大小的chunk同样按照最近使用顺序排列。ptmalloc使用“smallest-first，best-fit”原则在空闲large bins中查找合适的chunk。

当空闲的chunk被链接到bin中的时候，ptmalloc会把表示该chunk是否处于使用中的标志P设为0（注意，这个标志实际上处在下一个chunk中），同时ptmalloc还会检查它前后的chunk是否也是空闲的，如果是的话，ptmalloc会首先把它们合并为一个大的chunk，然后将合并后的chunk放到unstored bin中。要注意的是，并不是所有的chunk被释放后就立即被放到bin中。ptmalloc为了提高分配的速度，会把一些小的的chunk先放到一个叫做fast bins的容器内。

除了使用bins来管理内存外还有fast bins、top chunk、mmaped chunk、last remainder。

Fast bins：

Fast bins是bins的高速缓冲区，大约有10个定长队列。当用户释放一块不大于max\_fast的chunk时，会默认被放在fast bins上，当用户下次申请内存的时候会首先到fast bins上寻找是否有合适的chunk，然后才会到Bins上的空闲chunk.

Unsorted bin:

Unsorted bin是bins的一个缓冲区。当用户释放大于max\_fast的chunk时会进入unsorted bin。当用户malloc时会首先到unsorted bin中查找是否有合适的bin。

Top chunk：

并不是所有的chunk都会被放在bins上，top chunk相当于分配区的顶部空闲内存，当bins上都不能满足内存分配要求时，就会来top chunk上分配 。

Mmaped chunk:

当分配的内存非常大（大于128K）时，需要使用mmap,则会放到mmaped chunk，当释放mmaped chunk的内存直接归还给操作系统。

Malloc流程：

1. 获取分配区的锁，防止多线程冲突。

2. 计算出需要分配的内存的chunk实际大小。

3. 判断chunk的大小，如果小于max\_fast（64b），则取fast bins上去查询是否有适合的chunk，如果有则分配结束。

4. chunk大小是否小于512B，如果是，则从small bins上去查找chunk，如果有合适的，则分配结束。

5. 继续从 unsorted bins上查找。如果unsorted bins上只有一个chunk并且大于待分配的chunk，则进行切割，并且剩余的chunk继续扔回unsorted bins；如果unsorted bins上有大小和待分配chunk相等的，则返回，并从unsorted bins删除；如果unsorted bins中的某一chunk大小 属于small bins的范围，则放入small bins的头部；如果unsorted bins中的某一chunk大小 属于large bins的范围，则找到合适的位置放入。

6. 从large bins中查找，找到链表头后，反向遍历此链表，直到找到第一个大小 大于待分配的chunk，然后进行切割，如果有余下的，则放入unsorted bin中去，分配则结束。

7. 如果搜索fast bins和bins都没有找到合适的chunk，那么就需要操作top chunk来进行分配了（top chunk相当于分配区的剩余内存空间）。判断top chunk大小是否满足所需chunk的大小，如果是，则从top chunk中分出一块来。

8. 如果top chunk也不能满足需求，则需要扩大top chunk。主分区上，如果分配的内存小于分配阀值（默认128k），则直接使用brk()分配一块内存；如果分配的内存大于分配阀值，则需要mmap来分配；非主分区上，则直接使用mmap来分配一块内存。通过mmap分配的内存，就会放入mmap chunk上，mmap chunk上的内存会直接回收给操作系统。

Free流程：

1. 获取分配区的锁，保证线程安全。

2. 如果free的是空指针，则返回，什么都不做。

3. 判断当前chunk是否是mmap映射区域映射的内存，如果是，则直接munmap()释放这块内存。前面的已使用chunk的数据结构中，我们可以看到有M来标识是否是mmap映射的内存。

4. 判断chunk是否与top chunk相邻，如果相邻，则直接和top chunk合并（和top chunk相邻相当于和分配区中的空闲内存块相邻）。转到步骤8

5. 如果chunk的大小大于max\_fast（64b），则放入unsorted bin，并且检查是否有合并，有合并情况并且和top chunk相邻，则转到步骤8；没有合并情况则free。

6. 如果chunk的大小小于 max\_fast（64b），则直接放入fast bin，fast bin并没有改变chunk的状态。没有合并情况，则free；有合并情况，转到步骤7

7. 在fast bin，如果当前chunk的下一个chunk也是空闲的，则将这两个chunk合并，放入unsorted bin上面。合并后的大小如果大于64KB，会触发进行fast bins的合并操作，fast bins中的chunk将被遍历，并与相邻的空闲chunk进行合并，合并后的chunk会被放到unsorted bin中，fast bin会变为空。合并后的chunk和topchunk相邻，则会合并到topchunk中。转到步骤8

8. 判断top chunk的大小是否大于mmap收缩阈值（默认为128KB），如果是的话，对于主分配区，则会试图归还top chunk中的一部分给操作系统。free结束。