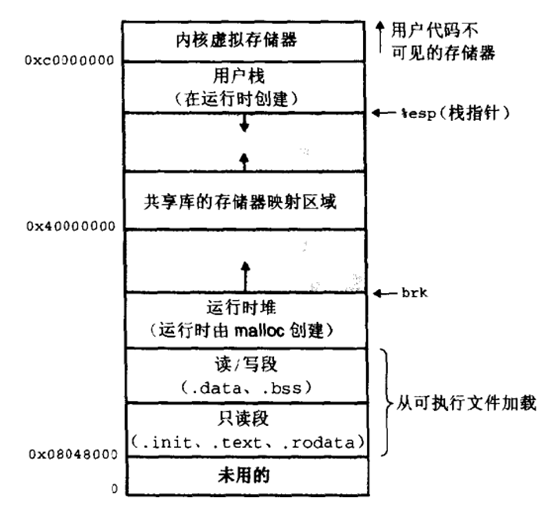
Linux操作系统存储器管理

在了解Linux的存储器管理之前，首先要知道对于一个进程来说它都会涉及到5种不同的数据段。稍有编程知识的朋友都能想到这几个数据段中包含有“**程序代码段**”、“**程序数据段**”、“**程序堆栈段**”等。这几种数据段都在其中，但除了以上几种数据段之外，进程还另外包含两种数据段。下面来简单归纳一下进程对应的内存空间中所包含的5种不同的数据区。

* 代码段：代码段是用来**存放可执行文件的操作指令**，也就是说是它是可执行程序在内存中的镜像。代码段需要**防止在运行时被非法修改**，所以只准许读取操作，而不允许写入（修改）操作——它是不可写的。
* 数据段：数据段用来存放可执行文件中已初始化全局变量，换句话说就是存放程序静态分配的变量和全局变量。
* BSS段：BSS段包含了程序中**未初始化的全局变量**，在内存中 bss段全部置零。
* 堆（heap）：堆是用于**存放进程运行中被动态分配的内存段**，它的大小并不固定，可动态扩张或缩减。当进程调用malloc等函数分配内存时，新分配的内存就被动态添加到堆上（堆被扩张）；当利用free等函数释放内存时，被释放的内存从堆中被剔除（堆被缩减）
* 栈：栈是**用户存放程序临时创建的局部变量**，也就是说我们函数括弧“{}”中定义的变量（但不包括static声明的变量，static意味着在数据段中存放变量）。除此以外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放回栈中。由于栈的先进先出特点，所以栈特别方便用来保存/恢复调用现场。从这个意义上讲，我们可以把堆栈看成一个寄存、交换临时数据的内存区。

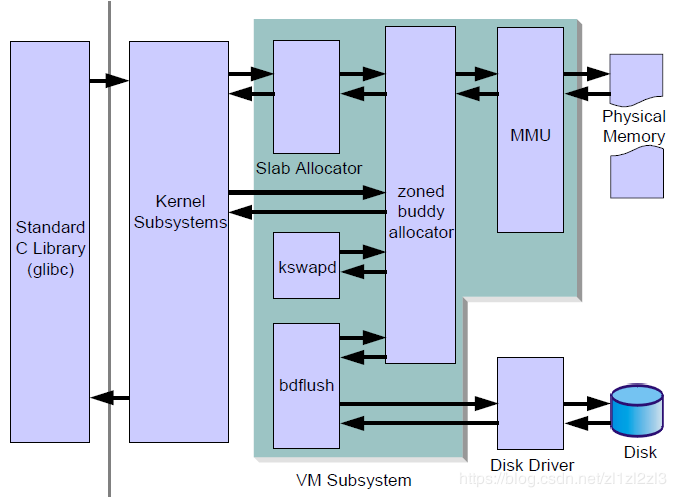
这五个数据段的组织形式如下图所示（引自：User-Level Memory Management）：



这就是Linux系统中进程的逻辑内存分布，从用户向内核看，所使用的内存表象形式会依次经历“**逻辑地址**”—“**线性地址**”—“**物理地址**”几种形式。逻辑地址经段机制转化成线性地址；线性地址又经过页机制转化为物理地址。（虽然逻辑地址和线性地址是两种不同的地址空间，但在Linux中逻辑地址就等于线性地址，它们的值是一样的，至于原因后面有讲述）。

Linux操作系统采用**虚拟内存管理技术**，使得每个进程都有各自互不干涉的进程地址空间。该空间是块大小为4G的线性虚拟空间**，用户所看到和接触到的都是该虚拟地址**，无法看到实际的物理内存地址。利用这种虚拟地址不但能起到**保护操作系统的效果**（用户不能直接访问物理内存），而且更重要的是，用户程序可使用比实际物理内存更大的地址空间。

为了更好地理解linux的虚拟存储技术，先放一张Linux虚拟存储的架构图：



MMU的全称是Memory Manager Unit，它与Physical Memory交换信息，zoned buddy allocator就是操作系统教课书中讲述的**伙伴系统算法**的实现手段，在将其称之为：**划区域的伙伴分配器**，而上面有一个slab allocator依然是一个关于内存的分配器，我们所知道的是当伙伴算法将一组相同内存大小的块连成一个链表供进程使用，很多时候容易出现**内存碎片，**就是这个内存块的大小太小了，因此就有了这个slab allocator来处理这些内存碎片。slab allocator的基础是 Jeff Bonwick 为 SunOS 操作系统首次引入的一种算法。kswapd 是一个 daemon 进程, 对系统内存做定时检查, 一般是1秒一次. 如果发现没有足够的空闲页面, 就做页回收(page reclaiming), 将不再使用的页面换出. 如果要换出的页面脏了, 往往还需要写回到磁盘或者到交换区。

MMU的主要职责就是将逻辑地址, 转成物理地址. 以32位Linux为例, MMU可以当成一个数学函数: f(x) = y, 输入x是一个0-4G范围内的逻辑地址, 输出y是实际的物理地址。

实际上并不需要把所有的映射关系都建立起来, 而只需要为用到的内存做映射, 所以, Linux用了分页的方法来解决这个问题(实际是一个多阶哈希).

一个典型的二级页表来处理分页: 32位的逻辑地址被分成了3段, 10位的一级页表索引(page table 1 index), 10位的二级页表索引(page table 2 index)和剩下的12位页面偏移量(page offset). 所谓页面(page), 是现在大部分MMU中用来管理内存的单位, Linux下常见的page大小是4k(12位的偏移量刚好是一个page, 即4k)。

在多级页表中, 页表分级越多, 越灵活, 但是带来的时间成本也就越高, 复杂度也越高. 二级, 或者三级页表是一个比较合理的选择. 为了兼容不同的CPU, Linux 2.6.11 之后使用了四级分页机制, 在不同的CPU环境下可以灵活扩展成二级或者三级。

当一个逻辑地址, 经过MMU映射后发现, 对应的页表项还没有映射到物理内存, 就会触发缺页错误(page fault): CPU必须找到一个可用的物理内存页面, 从页表项映射过去. 如果这个时候没有空闲的物理内存页面, 就需要做页面置换了, 操作系统通过某些算法, 从物理内存中选一个当前在用的页面, (是否需要写到磁盘, 取决于有没有被修改过), 重新调入, 建立页表项到之的映射关系。

Linux下采用的是**段页式内存管理,** 先分段, 再分页. 但是因为Linux中所有的段基址都设置成了0, 段偏移量相当于就是线性地址, 只用了一个地址空间, 效果上就是正常的分页. 这么做的原因是**为了兼容各种硬件体系**。

虽然Linux下, **"分段"只是一个摆设**, 但是在进程的内存管理中, 还是应用了分段的思想的: 每一个进程在运行时, 它的逻辑地址空间都会被分为代码段, 数据段, 堆, 栈等, 当访问段之外的内存地址时, kernel SubSystem能监测到并给出段错误(segment fault)。

最后来做一个简单的总结，在Linux上程序如果需要使用内存使用standard C library来调用Linux提供接口来访问，通过Kernel subsystem的处理，对程序进行分段，但在Linux中这只是个摆设，之后当需要的空间很大时直接调用zoned buddy allocator或者需要的是小内存时，先进入到slab allocator之后再进入buddy allocator中，从disk中读取到或写入数据时首先经过kswapd检验是否为脏数据，如果不是进行下一步的处理之后就是课本上所提到的分页管理方式了，通过MMU实现。

参考文献：

[1] 陈科.Linux内核分析及应用.北京：机械工业出版社[M], 2018-8-1.

[2] 汤小丹，梁红兵，哲凤屏，汤子瀛.计算机操作系统第四版[M].西安:西安电子科技大学出版社, 2014-05.

[3] 刘振洪，吴敏凤．Linux操作系统实用教程．天津:天津科学技术出版社，2016：4-20

[4] 赤兔欢. Linux 内存管理.https://www.jianshu.com/p/eecbb1506eee, 2017-03-19.

[5] M. Jones. Linux slab 分配器剖析. https://www.ibm.com/developerworks/cn/linux/l-linux-slab-allocator/:2010-9-20.