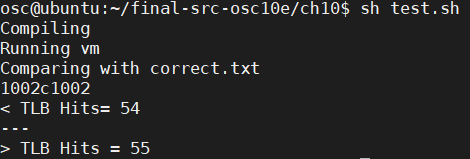
**OS Assignment 02 实验报告**

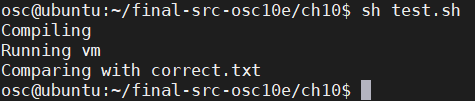
1. **虚存管理模拟实现**
2. MemorySize=256，fifo

结果截图：



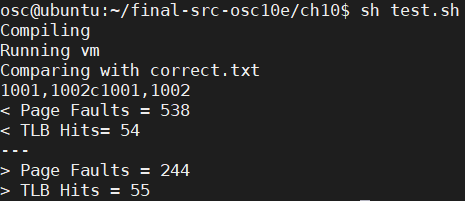
映射内容与correct完全一致，tlb hit=55，page fault=244

1. MemorySize=256，lru tlb



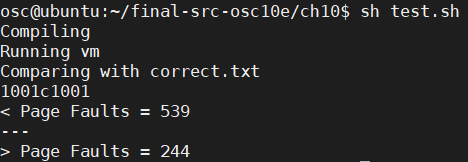
结果与correct完全一致，tlb hit=54，page fault=244

1. MemorySize=128，fifo策略



映射内容与correct完全一致，tlb hit=54，page fault=538

1. MemorySize=128，lru策略

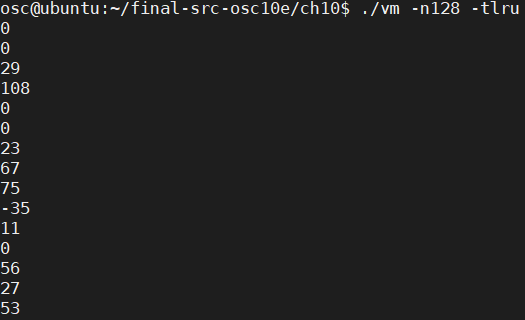


映射内容与correct完全一致，tlb hit=55，page fault=539

*说明：test.sh文件中设置的是物理内存大小为256，fifo策略的运行方式。查看其他物理内存和策略运行结果，可以用命令行：*

*./vm -n(指定内存大小） -t（fifo/lru）*

*例如：下图采用128字节物理内存和lru策略运行，得到输出结果*

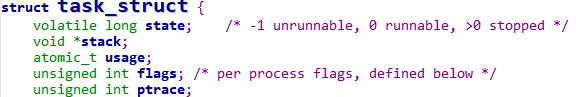


**2.Linux内存管理实验**

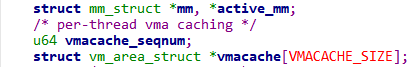
**（1）分析图1（注：图1是2级页表，对应于IA-32位系统），解释图中每一类方框和箭头的含义，在代码树中寻找相关数据结构片段，做简单解释。**

从图片底部开始

**task\_struct：**

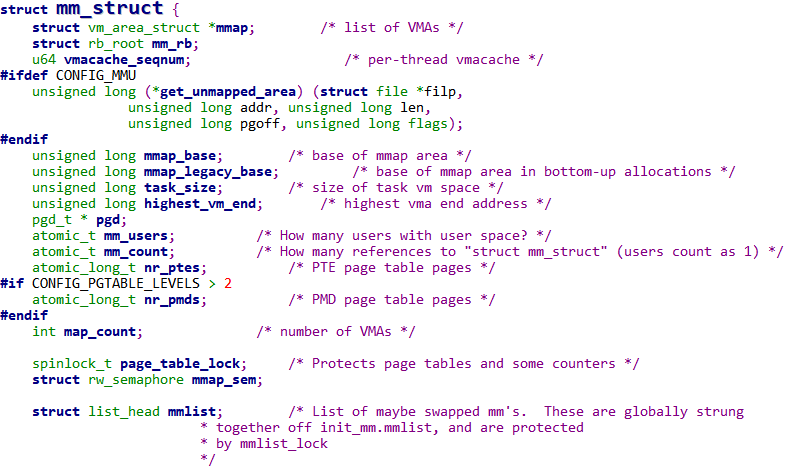


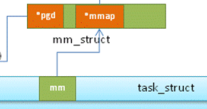
……

//内存描述符指针

在进程结构体中与内存管理有关的成员有内存描述符指针mm即下图的绿色区域，每个进程只有一个mm\_struct结构，在每个进程的task\_struct结构中，有一个指向该进程的结构。可以说，mm\_struct结构是对整个用户空间的描述。

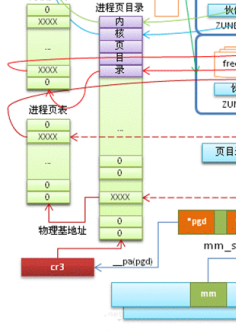
在源码中找到mm\_struct的定义如下，

**mm\_struct：**



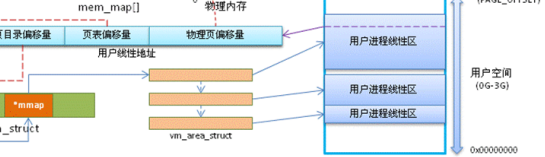
图中提示我们关注\*pgd和\*mmap两个指针。

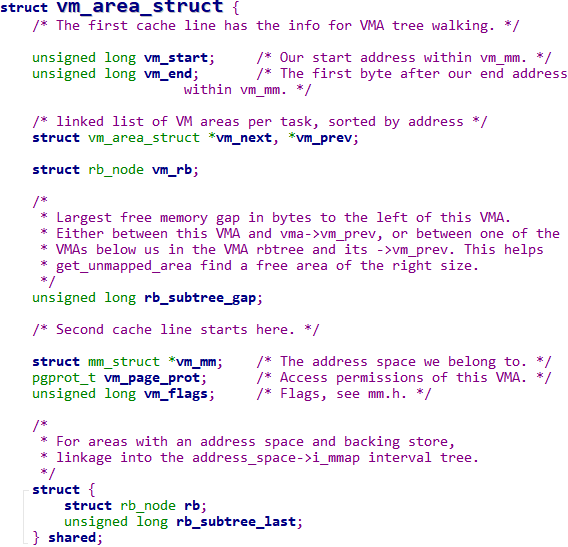
**pgd指针**：每个进程都有一个指向其自己的PGD指针，它是一个物理页面帧，包括一个pgd\_t类型的结构体，实际上以一个无符号长整形的树，将它复制到cr3寄存器即可实现进程页表的载入如下图蓝色箭头——红色箭头。





**mmap指针**：它指向的是vm\_area\_struct结构体，表示虚拟的进程地址空间，vm\_area\_struct描述了这个区间的内存范围、内存区域的类型等等。不相邻的区间由vm\_area\_struct\*vm\_next,vm\_prev连接形成形成VMA虚拟内存区域链表。它们共用0~3G连续的的虚拟内存用户空间，如下图。



**Vm\_area\_struct：**

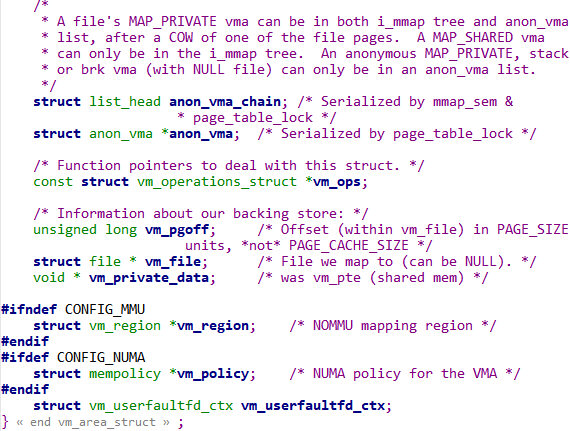
//区域的首尾地址

//VMA链表

//VMA的红黑树结构

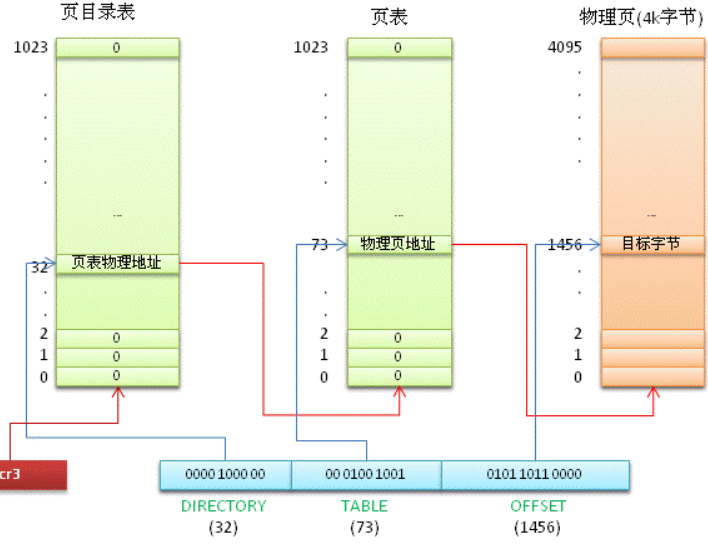
//访问控制权限

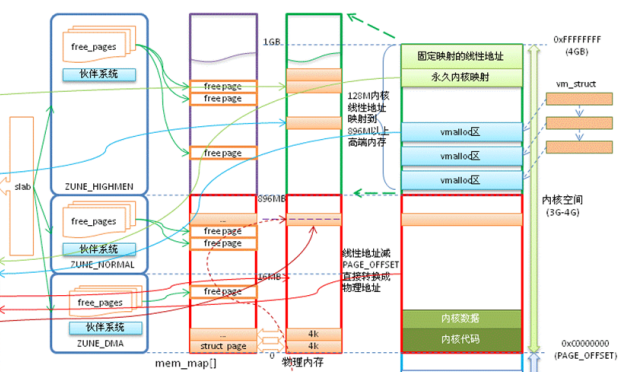
//保护标志位



//相关的操作表

用户进程线性区要想映射到物理内存需要通过用户线性地址转换，用户线性地址包括高10位页目录表偏移量，中间10位页表偏移量，低12位为物理页内的字节偏移量。首先根据页目录偏移量在页目录中查找页表物理地址，再根据此页表物理地址和中间10位的页表偏移量在页表中查找物理页地址，物理页地址+低12位的物理页内偏移量即是最终的物理地址，去该物理页中找到线性地址的实际内容。这也即是图中下面这一块的含义。



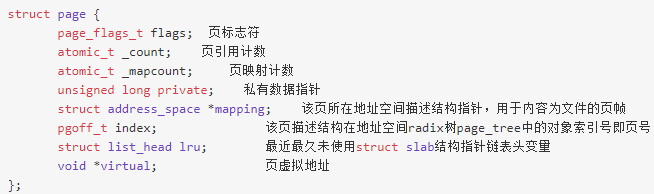
虚拟进程地址空间大小为4G，除了3G的用户进程空间，剩余3G~4G为内核空间，只有用户进程进行系统调用(代表用户进程在内核态执行)等情况才可访问。对应下图来看，内核空间又分为896M物理页面映射区（右侧长方形下部红框区域）、8M安全保护区域和120M内核地址空间（右侧长方形上方绿框区域）。其中896M的物理页面映射区，包含了内核镜像、物理页框表mem\_map等等，连续映射（线性地址减PAGE\_OFFSET即为物理地址）到ZONE\_DMA和ZONE\_NORMAL（左侧长方形下面两部分）。

剩下的ZONE\_HIGHMEN由用户进程和内核程序共享。内核程序对应的vm\_struct通过vmolloc申请分配一段虚拟地址空间，要对这段内存进行访问时同样需要将线性地址转换为页目录偏移、页表偏移和物理内存偏移量，分别去内核页目录、内核页表、物理内存中寻址。内核空间是由内核负责映射，不会跟着进程变化；它的页目录和进程页目录存在一块区域，但有其独立的内核页表。它也可以通过pkmap做临时映射或fixmap做固定地址映射。

无论是物理页面映射区还是内核地址空间的内存分配都应用了伙伴系统和SLAB两项机制。

伙伴系统以页框为单位分配内存，很好地解决了外部碎片问题。下面给出page结构体的定义

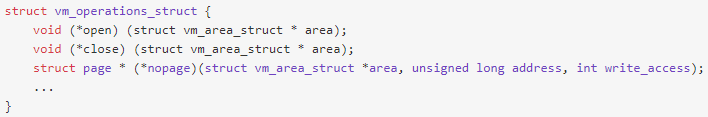
**Page：**



* flag页标志符含是不是脏的，是否被锁定等等，每一位单独表示一种状态，可同时表示出32种不同状态，定义在<linux/page-flags.h>
* \_count：计数值为-1表示未被使用。
* virtual：页在虚拟内存中的地址，对于不能永久映射到内核空间的内存(比如高端内存)，该值为NULL；需要事必须动态映射这些内存。

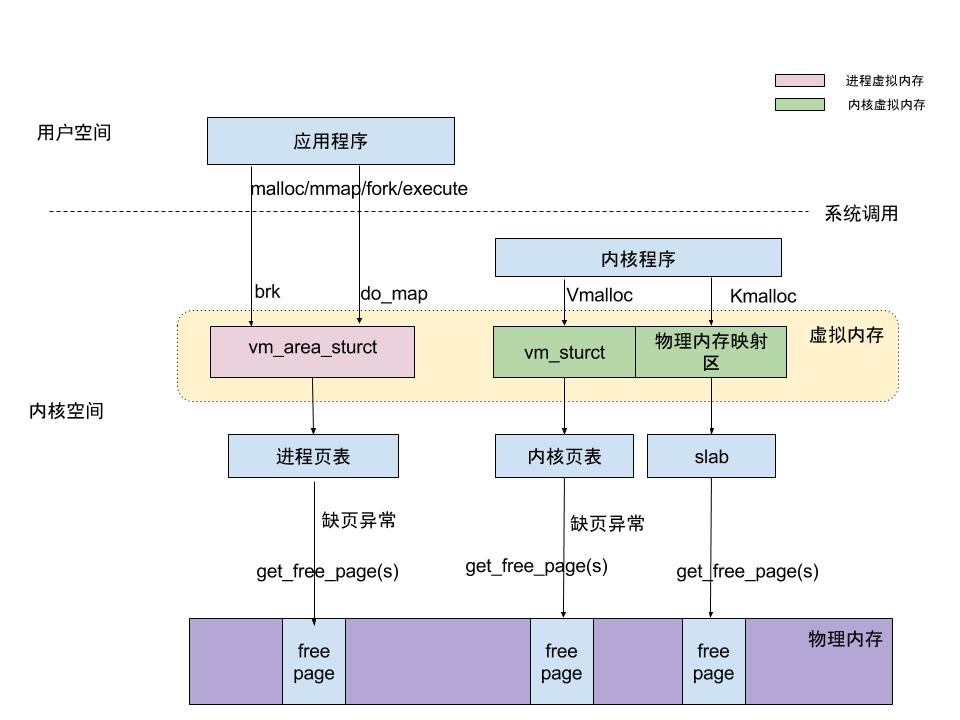
Slab机制针对经常分配和释放的小对象，如进程描述符这样的结构体，进行管理，相同类型的对象归为一类，每当要申请这样一个对象，slab分配器就从一个slab列表中分配一个这样大小的单元出去，而当要释放时，将其重新保存在该列表中，而不是直接返回给伙伴系统，从而避免这些内碎片。slab分配器并不丢弃已分配的对象，而是释放并把它们保存在内存中。当以后又要请求新的对象时，就可以从内存直接获取而不用重复初始化。 提高了效率也解决了内部碎片问题。

**Vm\_operations\_struct：**

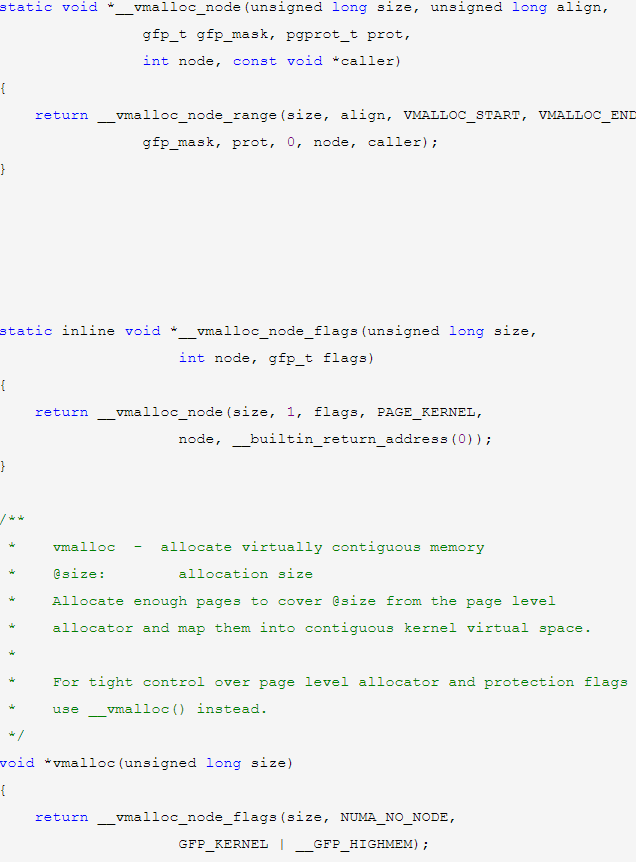


vm\_operations结构中包含的是函数指针；其中，open、close分别用于虚拟区间的打开、关闭，而nopage用于当虚存页面不在物理内存而引起的“缺页异常”时所应该调用的函数。

1. **参考图2解释内核层不同内存分配接口的区别，包括\_\_get\_free\_pages，kmalloc，vmalloc等**

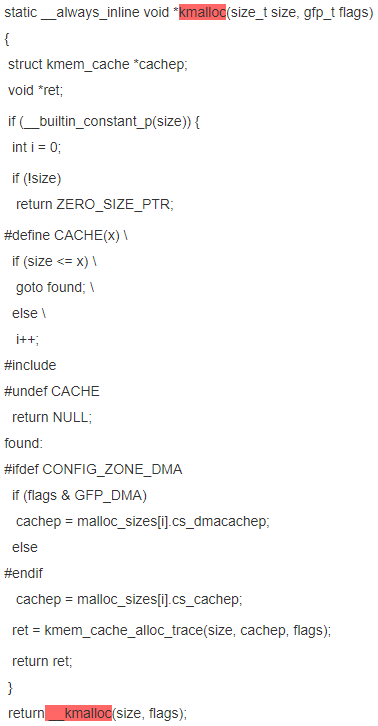


**Vmalloc:**



该函数返回的是一个指向内存块的指针，其内存块大小至少为size,所分配的内存是逻辑上连续的，以vm\_struct形式存在于128k的内核地址空间。

**Kmalloc:**



该函数返回的是一个指向内存块的指针，其内存块大小至少为size,所分配的内存在物理内存映射区中连续且保持原有的数据(不清零).其中部分flags取值说明：

GFP\_USER： 用于用户空间的分配内存，可能休眠；

GFP\_KERNEL：用于内核空间的内存分配，可能休眠；

GFP\_ATOMIC：用于原子性的内存分配，不会休眠；典型原子性场景有中断处理程序，软中断，tasklet等.

kmalloc分最多只能分配32个page大小的内存，每个page=4k，也就是128K大小，其中16个字节用来记录页描述结构。

**Get\_free\_pages:**



寻找mem\_map[0..(PAGING\_PAGES-1)]中的空闲项，即mem\_map[i]==0的项，如果找到，就返回物理地址，找不到返回0。所以说申请的内存是一整页，一页的大小一般是128K。

**三者的区别：**

①kmalloc和get\_free\_page申请的内存位于物理内存映射区域，而且在物理上也是连续的，它们与真实的物理地址只有一个固定的偏移，因此存在较简单的转换关系，而vmalloc仅在逻辑内存上连续，需要由页表实现较复杂的物理内存转化，开销较大，效率没有前两者高。

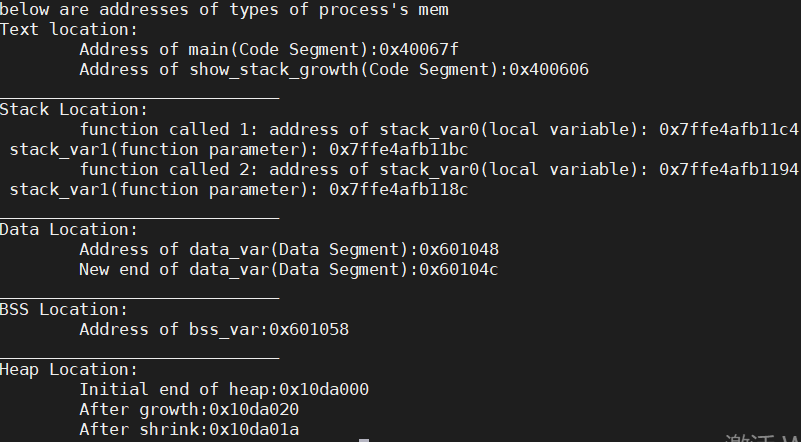
②vmalloc和get\_free\_page没有flag参数，默认可休眠，而kmalloc可以选择三种状态。

③kmalloc分配内存限制在128k-16M，get\_free\_page用于整页分配，而vmalloc可指定更大的size。

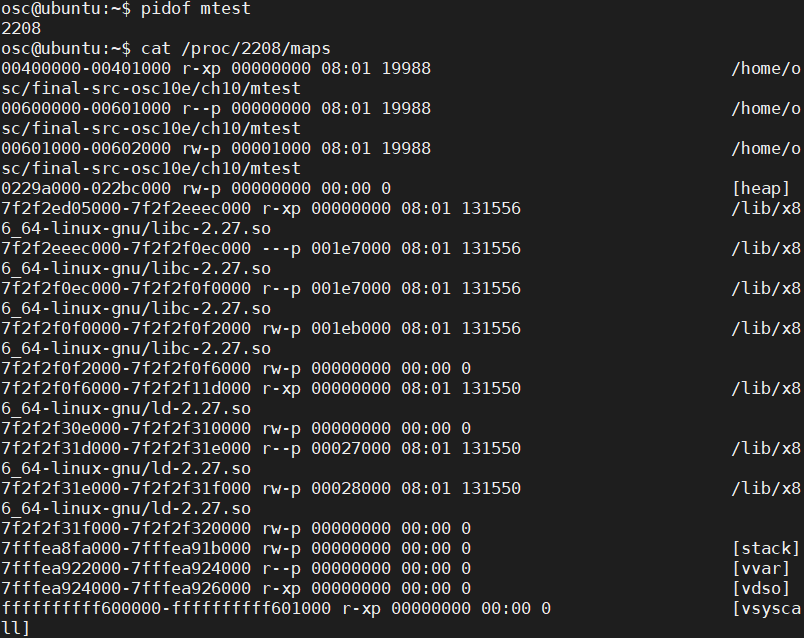
**三者的关系：**

kmalloc() 函数最终通过get\_free\_page实现，调用vmalloc（）分配到虚拟内存地址后若发生缺页异常，会调用get\_free\_page指定实际物理地址。

1. **参考[Anatomy of a Program in Memory](https://manybutfinite.com/post/anatomy-of-a-program-in-memory/)和[User-Level Memory Management](https://linuxdevices.org/ldfiles/misc/Linux_Programming_by_Example_ch03.pdf)中例程，写一个实验程序mtest.c，生成可执行程序mtest；打印代码段、数据段、BSS，栈、堆等的相关地址**



1. **参考How The Kernel Manages Your Memory，通过/proc/pid\_number/maps，分析mtest各个内存段（参考链接）。绘制图表，解释输出的每一段的各种属性，包括每一列的内容。为了让mtest程序驻留内存，可以在程序末尾加上长时睡眠，并将mtest在后台运行，即./mtest &**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 行  列 | 虚拟地址空间范围 | vm\_flags  （权限） | vm\_pgoff  （本段映射地址在文件中的偏移） | 文件所属设备号 | 文件所属节点号 | 文件名/角色 | 该行含义 |
| 1 | 00400000-00401000 | r -xp | 00000000 | 08:01 | 19988 | mtest | mtest程序代码段 |
| 2 | 00600000-00601000 | r --p | 00000000 | 08:01 | 19988 | mtest | mtest程序数据段 |
| 3 | 00601000-00602000 | rw-p | 00001000 | 08:01 | 19988 | mtest | mtest程序BSS段 |
| 4 | 0229a000-022bc000 | rw-p | 00000000 | 00:00 | 0 | heap | heap地址段 |
| 5 | 7f2f2ed05000-7f2f2eeec000 | r-xp | 00000000 | 08:01 | 131556 | libc-2.27.so | 内存映射段，用于分配内存区域 |
| 6 | 7f3eb20c6000-7f3eb22c6000 | ---p | 001e7000 | 08:01 | 131556 | libc-2.27.so |
| 7 | 7f3eb22c6000-7f3eb22ca000 | r--p | 001e7000 | 08:01 | 131556 | libc-2.27.so |
| 8 | 7f3eb22ca000-7f3eb22cc000 | rw-p | 001eb000 | 08:01 | 131556 | libc-2.27.so |
| 9 | 7f3eb22cc000-7f3eb22d0000 | rw-p | 00000000 | 00:00 | 0 |  |
| 10 | 7f3eb22d0000-7f3eb22f7000 | r-xp | 00000000 | 08:01 | 131550 | ld-2.27.so |
| 11 | 7f3eb24e8000-7f3eb24ea000 | rw-p | 00000000 | 00:00 | 0 |  |
| 12 | 7f3eb24f7000-7f3eb24f8000 | r--p | 00027000 | 08:01 | 131550 | ld-2.27.so |
| 13 | 7f3eb24f8000-7f3eb24f9000 | rw-p | 00028000 | 08:01 | 131550 | ld-2.27.so |
| 14 | 7f3eb24f9000-7f3eb24fa000 | rw-p | 00000000 | 00:00 | 0 |  |
| 15 | 7fff4f30f000-7fff4f330000 | rw-p | 00000000 | 00:00 | 0 | stack | 栈地址段 |
| 16 | 7fff4f335000-7fff4f337000 | r--p | 00000000 | 00:00 | 0 | vvar | 库函数数据映射区 |
| 17 | 7fff4f337000-7fff4f339000 | r-xp | 00000000 | 00:00 | 0 | vdso | 库函数映射区 |
| 18 | ffffffffff600000-ffffffffff601000 | r-xp | 00000000 | 00:00 | 0 | vsyscall | 固定地址映射区 |

1. **参考A Malloc Tutorial以及相关资料（如链接）**
2. 用户程序的内存分配涉及brk/sbrk和mmap两个系统调用，这两种方式的区别是什么，什么时候用brk/sbrk，什么时候用mmap？

答：brk/sbrk是在 向栈中压入数据容量超过栈的容量时 系统自动**扩大栈的大小**而使用的。也可以通过传递负参数缩小栈的大小。

Mmap用于将一个文件或其他对象**映射到**进程的虚拟地址空间**堆和栈中间**的地方，实现文件磁盘地址和进程虚拟地址一一对应的关系。他会在虚拟地址空间中找到一段连续空间，为此虚拟区分配一个vm\_area\_struct 结构，然后将其插入到进程的虚拟地址区的链表或树中，通过待映射的文件指针，在文件描述符表中找到对应的文件描述符，加入到struct file中，通过内核空间的系统调用函数mmap将用户空间与设备内存相连。之后建立页表，实现文件地址和虚拟地址区域的映射关系。

1. 应用程序开发时，为什么需要用标准库里的malloc而不是直接用这些系统调用接口？malloc额外做了哪些工作？

答：Malloc函数会根据申请的内存空间大小自动地选择使用brk系统调用或mmap系统调用。一般当申请小于128k的内存，使用brk分配内存，将栈空间往高地址推。malloc大于128k的内存，使用mmap分配内存，因为brk分配的内存需要等到高地址内存释放以后才能释放，导致产生内存碎片，而mmap分配的内存可以单独释放。

1. malloc的内存分配，是分配的虚拟内存还是物理内存？两者之间如何转换？

答：Malloc分配的是虚拟地址，通过建立页表和物理内存进行映射。