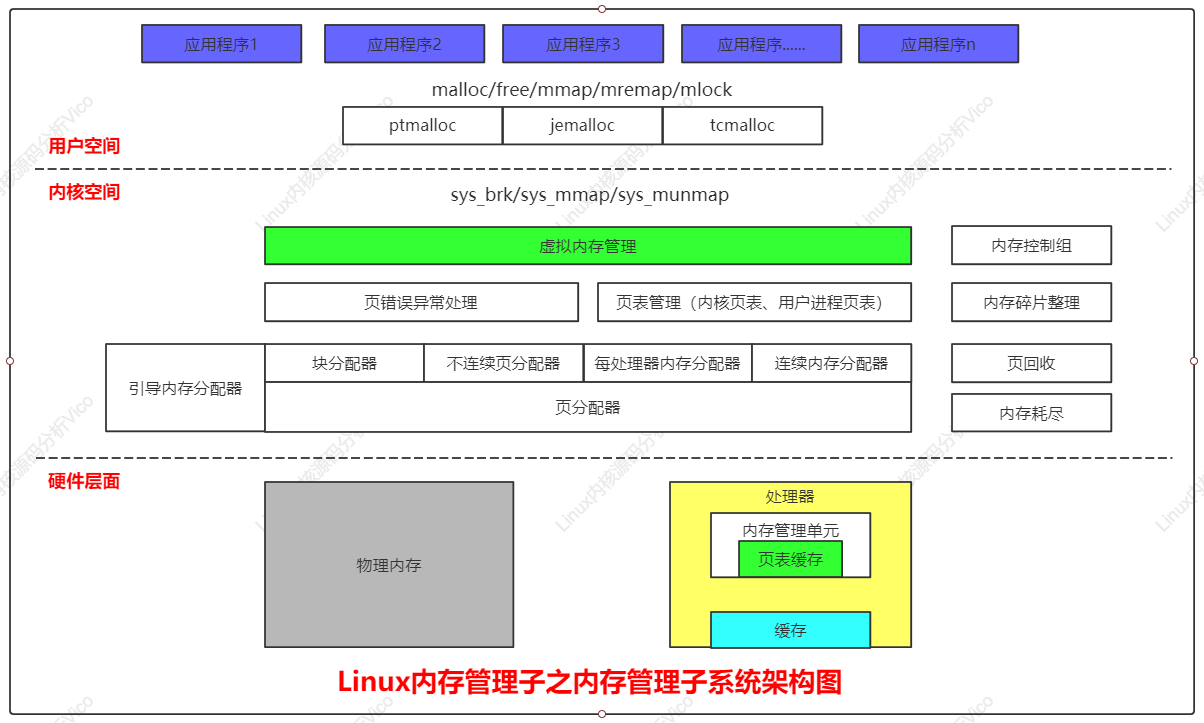
**Linux内核源码分析：内存管理子系统架构（用户空间、内核空间、硬件层面）**

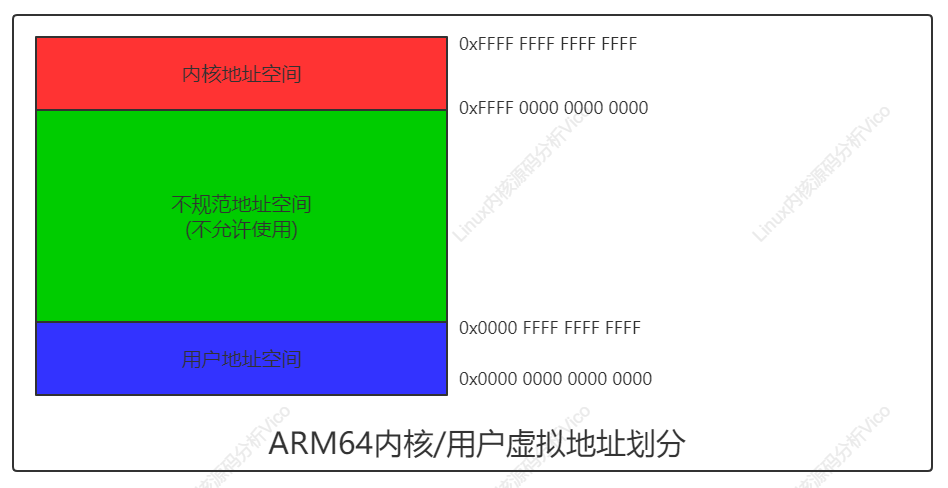


**一、虚拟地址布局/内存映射**

**1、虚拟地址空间划分**

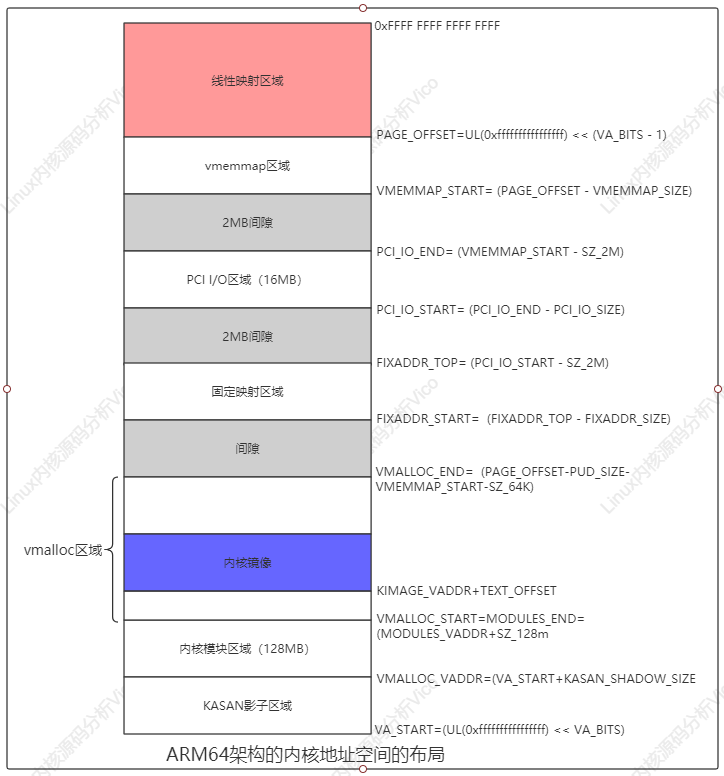
**1）用户虚拟地址空间布局**

目前应用程序没有那么大的内存需求、所以ARM64处理器不支持完全的64位虚拟地址，虚拟地址的最大宽度是48位，如下图所示：



**2）内核地址空间布局**

ARM64处理器架构的内核地址空间布局如下图所示：



**2、物理地址空间**

物理地址是处理器在系统总线上看到的地址。使用精简指令集（Reduced Instructions Set Computer，RISC）的处理顺路通常只一个物理地址空间，外围设备和物理内存使用统一的物理地址空间。有些处理顺路架构把分配给外围设备的物理地址区域称为设备内存。处理器通过外围设备控制器的寄存器访问外围设备，寄存器分为控制寄存器、状态寄存器和数据寄存器三大类，外围设备的寄存器通常被连续编址。处理器对外围设备寄存器的编址方式有两种：I/O映射方式（I/O-mapped）和内存映射方式（memory-mapped）。

程序只能通过虚拟地址访问外设寄存器，内核提供如下函数来把外设寄存器的物理地址映射到虚拟地址空间。

a.函数ioremap()把外设寄存器的物理地址映射到内核虚拟地址空间。

void \* ioremap(unsigned long phys\_addr, unsigned long size, unsigned long flags)

入口： phys\_addr：要映射的起始的IO地址；

[size](https://baike.baidu.com/item/size/2286120)：要映射的空间的大小；

flags：要映射的IO空间的和权限有关的标志；

phys\_addr：是要映射的物理地址；

size：是要映射的长度，单位是字节

b.函数io\_remap\_pfn\_range()把外设寄存器的物理地址映射到进程的用户虚拟地址空间。

int remap\_pfn\_range(struct vm\_area\_struct \*vma, unsigned long virt\_addr, unsigned long pfn, unsigned long size, pgprot\_t prot);

int io\_remap\_pfn\_range(struct vm\_area\_struct \*vma, unsigned long virt\_addr, unsigned long pfn, unsigned long size, pgprot\_t prot);

remap\_pfn\_range和io\_remap\_pfn\_range负责为一段物理地址建立新的页表，他们的原型如下函数的返回值是0或者是个负的错误码。

第一个函数是在pfn指向实际的RAM的时候使用，后面一个函数实在phy\_addr指向I/O内存时候使用，在实际使用中，除了SPARC外，对每个体系架构这两个函数都是等价的。大多数情况下使用第一个函数，对于有移植性的要求的驱动程序，要使用与特定情形相符的remap\_pfn\_range函数变种。

该函数的功能是创建页表。其中参数vma是内核根据用户的请求自己填写的，而参数addr表示内存映射开始处的虚拟地址，因此，该函数为addr~addr+size之间的虚拟地址构造页表。   另外，pfn（Page Fram Number）是虚拟地址应该映射到的物理地址的页面号，实际上就是物理地址右移PAGE\_SHIFT位。如果PAGE\_SHIFT为4kb，则PAGE\_SHIFT为12，因为PAGE\_SHIFT等于1<<PAGE\_SHIFT。最一个参数prot是新页所要求的保护属性。   在驱动程序中，一般能使用remap\_pfn\_range（）映射内存中的保留页（如X86系统中的640KB~1MB区域）和设备I/O内存。因此，如果想把kmalloc()申请的内存映射到用户空间，则可以通过mem\_map\_reserve()把相应的内存设置为保留后就可以。

c.内核提供函数iounmap()，它用来删除函数ioremap()创建的映射。

void \* \_\_ioremap(unsigned long phys\_addr, unsigned long size, unsigned long flags)

void \*ioremap(unsigned long phys\_addr, unsigned long size)

入口： phys\_addr：要映射的起始的IO地址；

[size](https://baike.baidu.com/item/size/2286120)：要映射的空间的大小；

flags：要映射的IO空间的和权限有关的标志；

phys\_addr：是要映射的物理地址

size：是要映射的长度，单位是字节

**3、内存映射应用编程接口实战**

内存映射是在进程的虚拟地址空间中创建一个映射，可分为两种：文件映射和匿名映射。通常把文件映射的物理页称为文件页，把匿名映射的物理页称为匿名页。

根据修改是否对其他进程 可见和是否传递到底层文件，内存映射分为共享映射和私有映射，两个进程可以使用共享的文件映射实现共享内存。匿名映射通常是私有映射，共享的匿名映射只可能出现在父进程和子进程之间。

在进程的虚拟地址空间中，代码段和数据段是私有的文件映射，未初始化数据段、堆和栈是私有的匿名映射。

**《内存映射的原理详解》**

**应用编程接口-->内存管理子系统提供如下常用的系统调用：**

a.mmap()来创建内存映射。

**void \*mmap(void \****addr***, size\_t** *length***, int** *prot***, int** *flags***,** **int** *fd***, off\_t** *offset***);**

b.mremap()来扩大或缩小已经存在的内存映射，可能同时移动。

**void \* mremap(void \****old\_address***, size\_t***old\_size* **, size\_t***new\_size***, int***flags***);**

c.munmap()用来删除内存映射。

**int munmap(void \****addr***, size\_t** *length***);**

d.brk()用来设备堆的上界。

**int brk(void \****addr***);**

e.remap\_file\_pages()用来创建非线性的文件映射，即文件区间和虚拟地址空间之间的映射不是线性关系，现在已被废除。

**int remap\_file\_pages(void \****addr***, size\_t** *size***, int** *prot***,size\_t** *pgoff***, int** *flags***);**

f.mprotect()用来设置虚拟内存区域的访问权限。

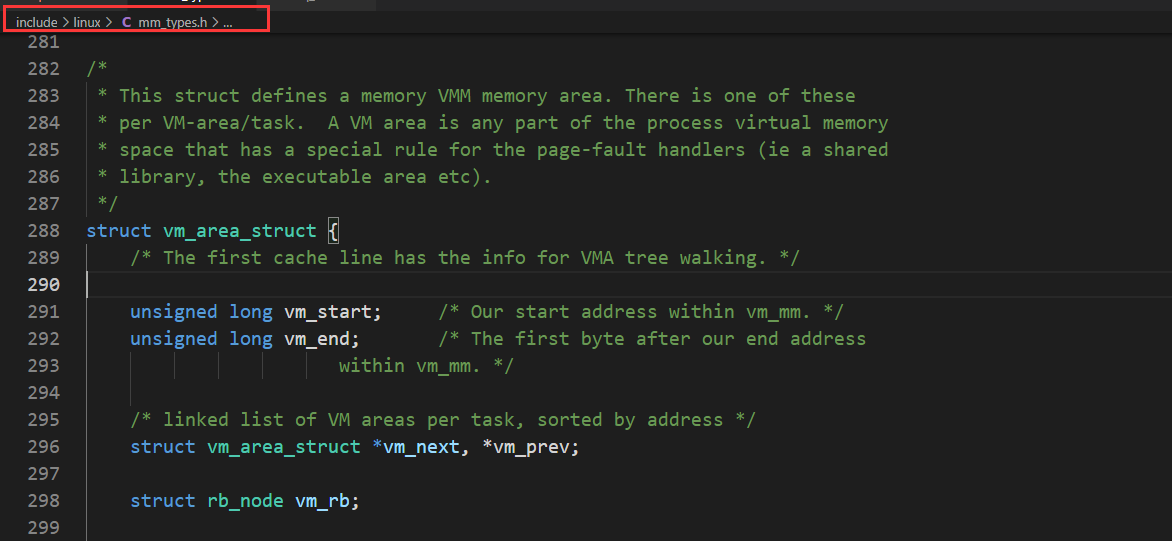
**int mprotect(void \****addr***, size\_t** *len***, int** *prot***);**

g.madvise()用来向内核提供提出内存使用的建议，应用程序告诉内核期望怎样使用指定的虚拟内存区域，以便内核可以选择适合的预读和缓存技术。

**int madvise(void \****addr***, size\_t** *length***, int** *advice***);**

**4、内存映射数据结构**

1）虚拟内存区域：是分配给进程的一个虚拟地址范围，内核使用结构体vm\_area\_struct描述虚拟内存区域：



**5、创建内存映射**

C标准库封装了函数mmap用来创建内存映射，内核提供POSIX标准定义的系统调用mmap如下：

6、虚拟内存提交策略

7、删除内存映射

**二、内存模型/引导内存分配器**

1、体系结构（NUMA、SMP）

2、内存模型（平坦内存、不连续内存、稀疏内存）

3、三级结构（node->zone->page）

4、bootmem分配器

5、memblock分配器

6、物理内存数据信息

**三、伙伴分配器/块分配器**

1、伙伴分配器基本知识

2、分区的伙伴分配器（数据结构，首选/备用区域类型、区域水线）

3、每处理器页集合

4、分配页/释放页

5、块分配器编程接口实战

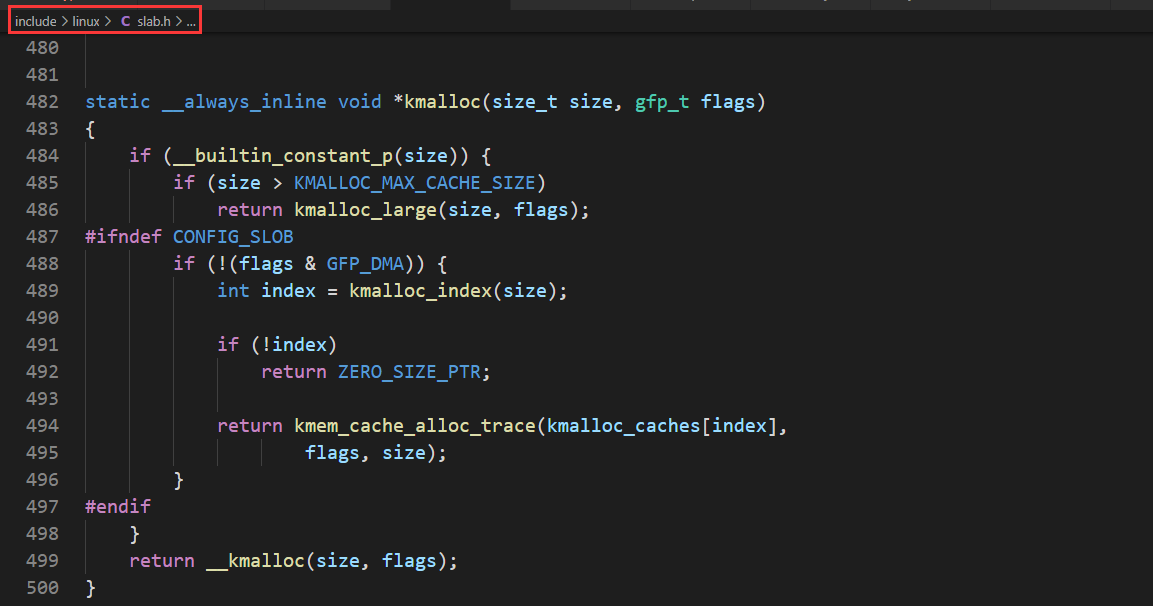
6、slab分配器、slub分配器、slob分配器

**一、kmalloc/vmalloc系统调用**

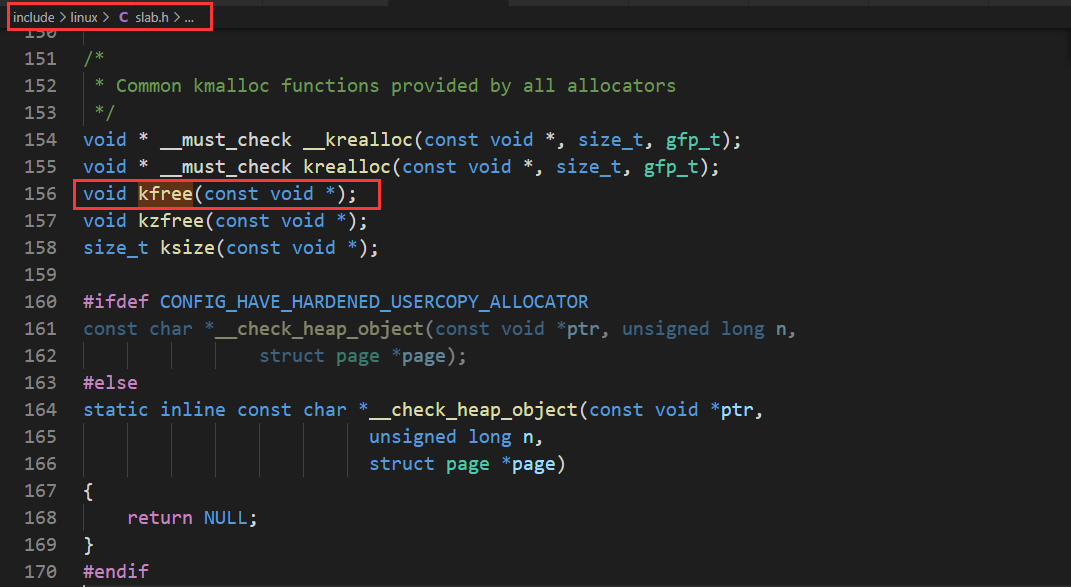
1、kmalloc()用于申请较小的/连续的物理内存，以字节为单位进行分配（在头文件</include/linux/slab.h>）麒麟操作系统（UbuntuKylin）20.00以上

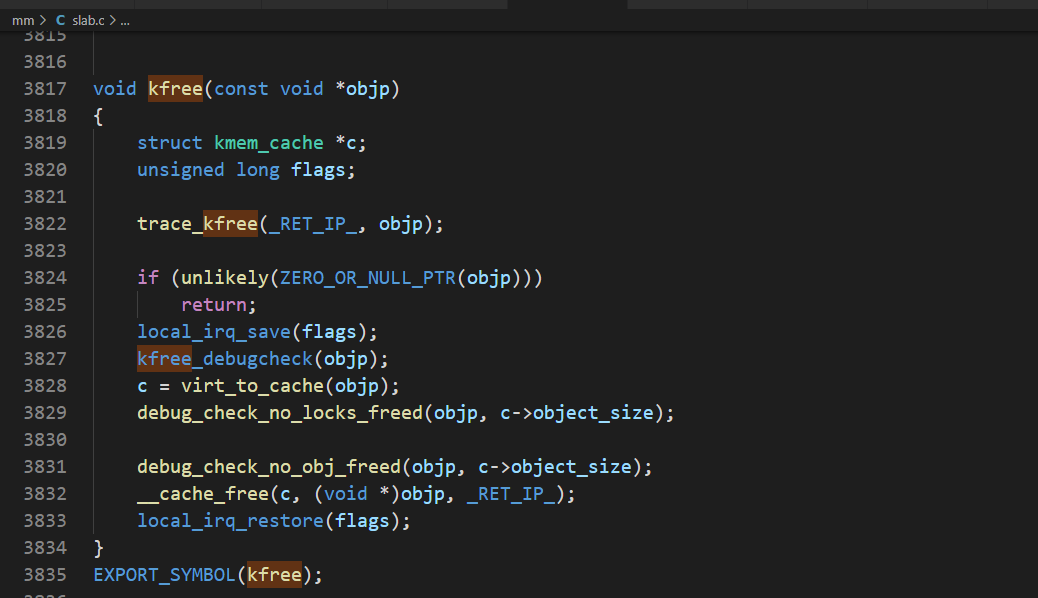
函数原型如下：

void \*kmalloc(size\_t size, int flags) // 分配的内存物理地址上连续，虚拟地址上自然连续



2、void kfree(const void \*ptr)释放由kmalloc()分配出来的内存块。

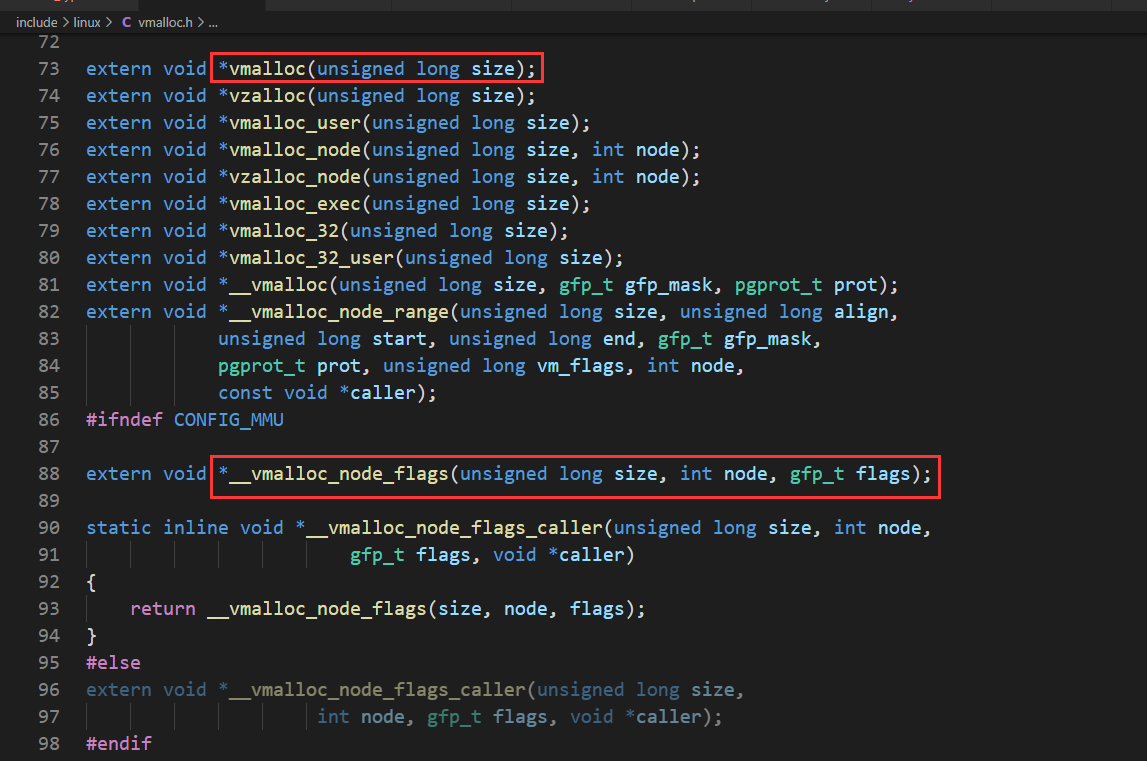


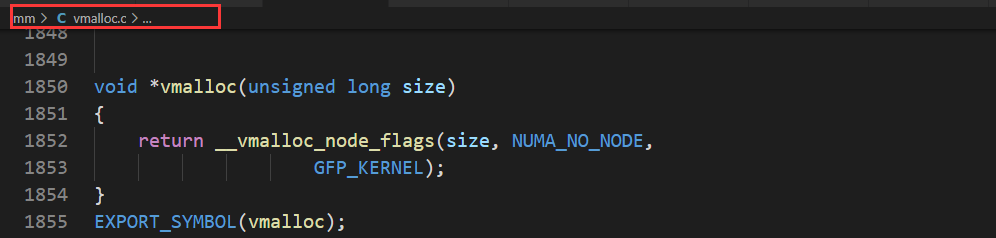


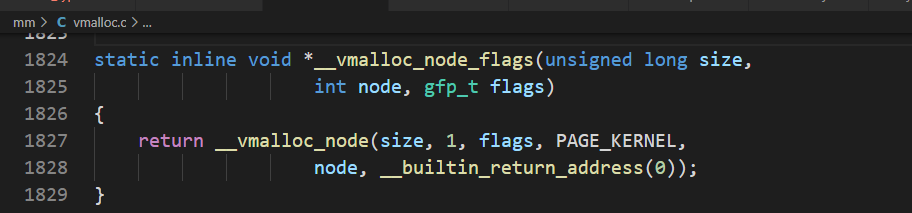
3、vmalloc()用于申请较大的内存空间，虚拟内存是连续的，以字节为单位进行分配（在头文件</include/linux/vmalloc.h>）

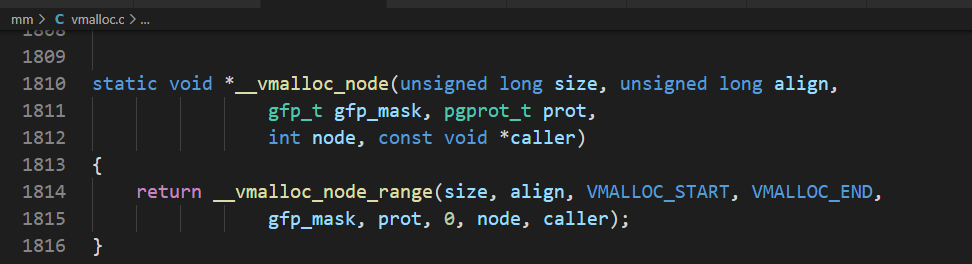
函数原型如下：

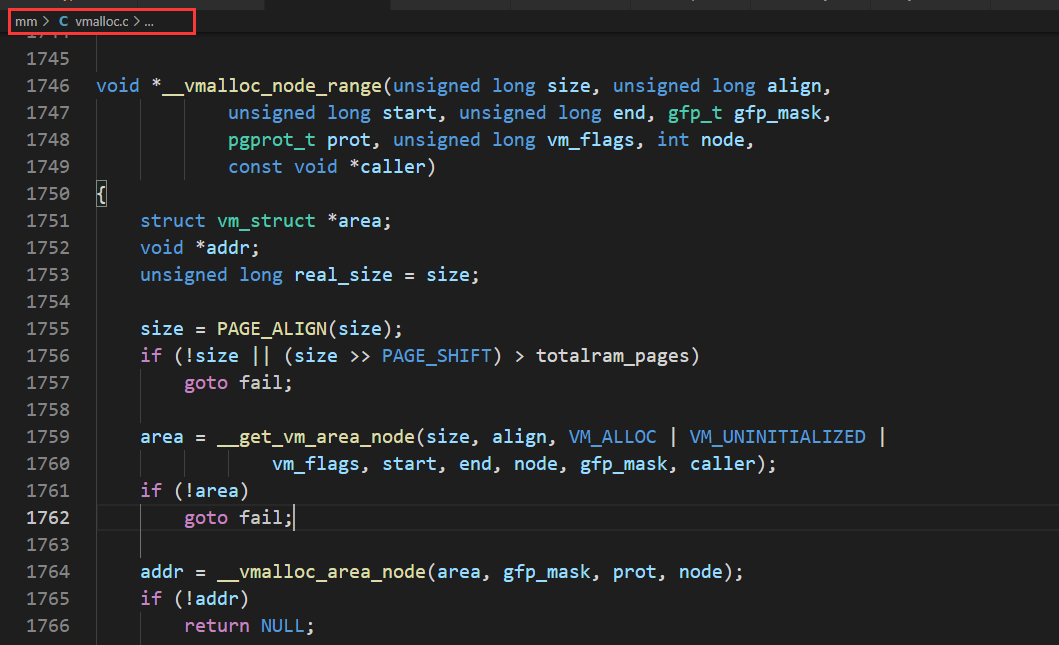
void \*vmalloc(unsigned long size) // 分配的内存虚拟地址上连续，物理地址不连续





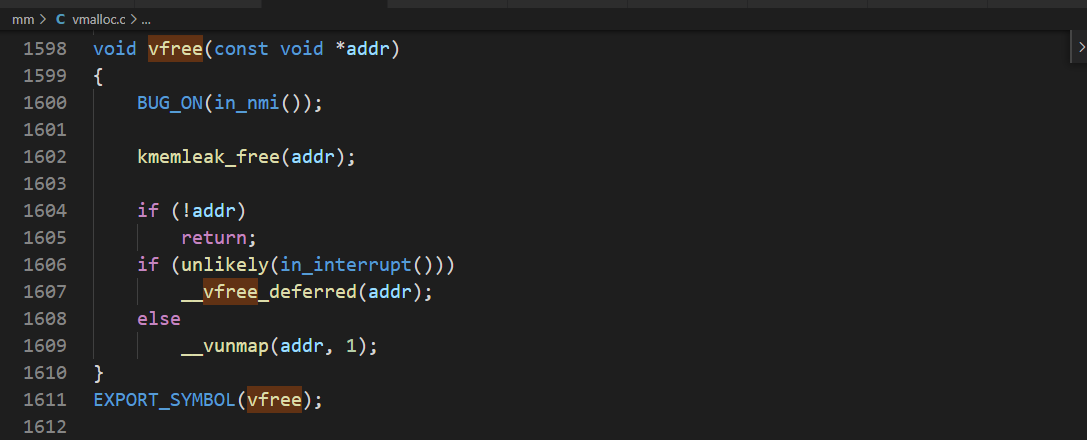






一般情况下，只有硬件设备才需要物理地址连续的内存，因为硬件设备往往存在于MMU之外，根本不了解虚拟地址；但为了性能上的考虑，内核中一般使用 kmalloc()，而只有在需要获得大块内存时才使用vmalloc()，例如当模块被动态加载到内核当中时，就把模块装载到由vmalloc()分配 的内存上。

4、void vfree(void \*addr) // 这个函数可以睡眠，因此不能从中断上下文调用。



5、malloc(), vmalloc()和kmalloc()区别

kmalloc和vmalloc是分配的是内核的内存，malloc分配的是用户的内存；

kmalloc保证分配的内存在物理上是连续的，vmalloc保证的是在虚拟地址空间上的连续，malloc不保证任何东西；

kmalloc能分配的大小有限，vmalloc和malloc能分配的大小相对较大；

内存只有在要被DMA访问的时候才需要物理上连续；

vmalloc比kmalloc要慢；

kmalloc对应于kfree，可以分配连续的物理内存；

vmalloc对应于vfree，分配连续的虚拟内存，但是物理上不一定连续。

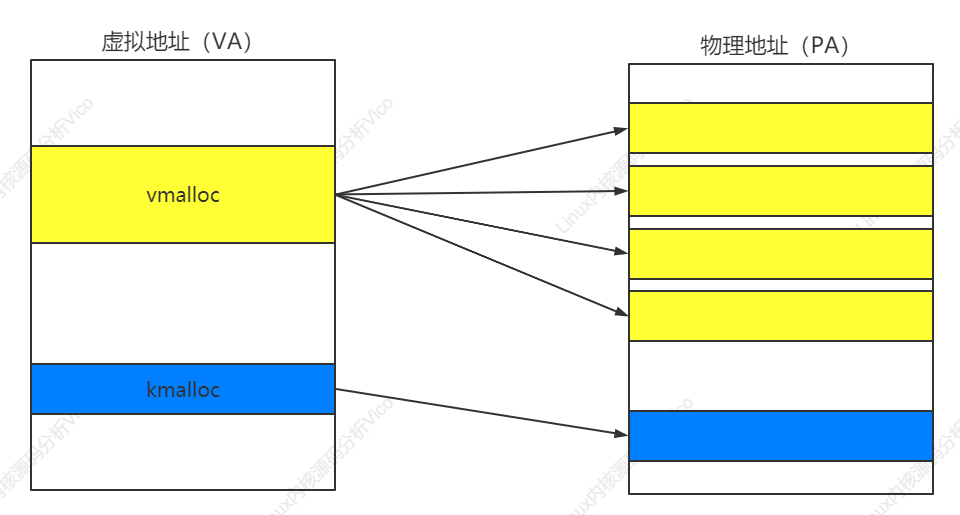
vmalloc分配内存的时候逻辑地址是连续的，但物理地址一般是不连续的，适用于那种一下需要分配大量内存的情况，如insert模块的时候。这种分配方式性能不入kmalloc。

kmalloc分配内存是基于slab，因此slab的一些特性包括着色，对齐等都具备，性能较好。物理地址和逻辑地址都是连续的。

6、最主要的区别是

在设备驱动程序或者内核模块中动态开辟内存，不是用malloc，而是kmalloc ,vmalloc，释放内存用的是kfree,vfree，kmalloc函数返回的是虚拟地址(线性地址). kmalloc特殊之处在于它分配的内存是物理上连续的,这对于要进行DMA的设备十分重要. 而用vmalloc分配的内存只是线性地址连续,物理地址不一定连续,不能直接用于DMA。vmalloc函数的工作方式类似于kmalloc，只不过前者分配的内存虚拟地址是连续的，而物理地址则无需连 续。通过vmalloc获得的页必须一个一个地进行映射，效率不高， 因此，只在不得已(一般是为了获得大块内存)时使用。vmalloc函数返回一个指针，指向逻辑上连续的一块内存区，其大小至少为size。在发生错误 时，函数返回NULL。vmalloc可能睡眠，因此，不能从中断上下文中进行调用，也不能从其它不允许阻塞的情况下调用。要释放通过vmalloc所获 得的内存，应使用vfree函数

vmalloc和kmalloc的分配内存结构图如下：



1：vmalloc分配的一般为高端内存，只有当内存不够的时候才分配低端内存；kmallco从低端内存分配；

2：vmalloc分配的物理地址一般不连续，而kmalloc分配的地址连续，两者分配的虚拟地址都是连续的；

3：vmalloc分配的一般为大块内存，而kmalloc一般分配的为小块内存，（一般不超过128k)。