```
Virtual kernel memory layout:

    vector  : 0xffff0000 - 0xffff1000   (   4 kB)

    fixmap  : 0xffc00000 - 0xffe00000   (2048 kB)

    vmalloc : 0xf0000000 - 0xff000000   ( 240 MB)

    lowmem  : 0xc0000000 - 0xef800000   ( 760 MB)

    pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)

    modules : 0xbf000000 - 0xbfe00000   (  14 MB)

      .text : 0xc0008000 - 0xc05cde80   (5912 kB)

      .init : 0xc05ce000 - 0xc0602000   ( 208 kB)

      .data : 0xc0602000 - 0xc0638728   ( 218 kB)

       .bss : 0xc0638728 - 0xc06a8af4   ( 449 kB)
```

kernel module被载入(load)到[0xbf000000, 0xbfe00000)的14M空间。

```
root@granite2:~# cat /proc/modules
galcore 160452 0 - Live 0xbf245000 (O)
ipv6 276100 20 [permanent], Live 0xbf1eb000
imagepower 1782 0 - Live 0xbf1e7000 (O)
stepper_api_a0 12747 0 - Live 0xbf1df000 (O)
upc 88464 0 - Live 0xbf1ab000 (O)
pegmatite_regulator 4051 4 imagepower,upc,[permanent], Live 0xbf1a3000
picdriver 55024 0 - Live 0xbf175000 (O)
cisxdriver 34852 0 - Live 0xbf153000 (O)
laservideo_a0 80001 0 - Live 0xbf135000 (O)
scanblkdriver 55179 0 - Live 0xbf11c000 (O)
icetestdriver 15129 0 - Live 0xbf113000 (O)
piedriver 179267 0 - Live 0xbf0d7000 (O)
dmaalloc 4127 1 laservideo_a0, Live 0xbf0cc000 (O)
dros 9786 2 laservideo_a0,scanblkdriver, Live 0xbf0c4000 (O)
hips_pll 4966 1 laservideo_a0, Live 0xbf0ac000 (O)
mv61_cdma 31142 0 - Live 0xbf09f000
sccplite 5695 0 - Live 0xbf093000 (O)
stepper_mod_a0 15653 1 stepper_api_a0, Live 0xbf05b000 (O)
ehci_hcd 44353 0 - Live 0xbf04b000
ci_hdrc_imx 2976 0 - Live 0xbf047000
usbmisc_imx 4330 1 ci_hdrc_imx, Live 0xbf042000
ci_hdrc 20768 1 ci_hdrc_imx, Live 0xbf038000
ipc_driver 4578 0 - Live 0xbf010000 (O)
i2c_pxa 8241 0 - Live 0xbf009000
```

dro_pegmatite 1598 0 - Live 0xbf005000

pegmatite_wdt 4332 0 - Live 0xbf000000

galcore 160452 0 - Live 0xbf245000 (O)

galcore载入的address为0xbf245000,size为160452 bytes.

当kernel载入一个module时，分配memory

kernel/module.c

load_module() --> move_module() --> module_alloc_update_bounds() --> module_alloc(size)

```
static void *module_alloc_update_bounds(unsigned long size)
{
        void *ret = module_alloc(size);

        if (ret) {
                mutex_lock(&module_mutex);
                /* Update module bounds. */
                if ((unsigned long)ret < module_addr_min)
                        module_addr_min = (unsigned long)ret;
                if ((unsigned long)ret + size > module_addr_max)
                        module_addr_max = (unsigned long)ret + size;
                mutex_unlock(&module_mutex);
        }
        return ret;
}
```

所有module占用的空间在[module_addr_min, module_addr_max)之间。

/* Bounds of module allocation, for speeding __module_address.

 * Protected by module_mutex. */

static unsigned long module_addr_min = -1UL, module_addr_max = 0;

查看以下这两个variables的值。

```
walterzh$ nm vmlinux-3.18.7-yocto-standard | grep module_addr_min

c061268c d module_addr_min

walterzh$ nm vmlinux-3.18.7-yocto-standard | grep module_addr_max

c068d288 b module_addr_max
```

```
root@granite2:~# devmem 0x0061268c

0xBF000000

root@granite2:~# devmem 0x0061268c     8d288

0xBF276C24
```

[0xBF000000， 0xBF276C24) 之间。

```c
void * __weak module_alloc(unsigned long size)
{
        return vmalloc_exec(size);
}
```

```
/**
 *      vmalloc_exec  -  allocate virtually contiguous, executable memory
 *      @size:          allocation size
 *
 *      Kernel-internal function to allocate enough pages to cover @size
 *      the page level allocator and map them into contiguous and
 *      executable kernel virtual space.
 *
 *      For tight control over page level allocator and protection flags
 *      use __vmalloc() instead.
 */

void *vmalloc_exec(unsigned long size)
{
        return __vmalloc_node(size, 1, GFP_KERNEL |__GFP_HIGHMEM, PAGE_KERNEL_EXEC,
                              NUMA_NO_NODE, __builtin_return_address(0));
}
```

module占用的空间优先从highmem分配。
并且由于module是code，所以占用的空间必须是executable。

```
static void *__vmalloc_node(unsigned long size, unsigned long align,
                            gfp_t gfp_mask, pgprot_t prot,
                            int node, const void *caller)
{
        return __vmalloc_node_range(size, align,VMALLOC_START, VMALLOC_END,
                                    gfp_mask, prot, node, caller);
}
```

#ifndef CONFIG_THUMB2_KERNEL

#define MODULES_VADDR   (PAGE_OFFSET - SZ_16M)

#else

/* smaller range for Thumb-2 symbols relocation (2^24)*/

#define MODULES_VADDR   (PAGE_OFFSET - SZ_8M)

#endif

/*

 * The highmem pkmap virtual space shares the end of the module area.

 */

```
#ifdef CONFIG_HIGHMEM

#define MODULES_END             (PAGE_OFFSET - PMD_SIZE)

#else

#define MODULES_END             (PAGE_OFFSET)

#endif


#define PMD_SHIFT               21

#define PMD_SIZE        (1UL << PMD_SHIFT)
```

MODULES_VADDR = 0xc0000000 - 16M = 0xbf000000

MODULES_END = 0xc0000000 - 1UL << 21 = 0xc0000000 - 2M = 0xbfe00000

We could get the verification from the following boot log.

```
Virtual kernel memory layout:

    vector  : 0xffff0000 - 0xffff1000   (   4 kB)

    fixmap  : 0xffc00000 - 0xffe00000   (2048 kB)

    vmalloc : 0xf0000000 - 0xff000000   ( 240 MB)

    lowmem  : 0xc0000000 - 0xef800000   ( 760 MB)

    pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
```

```
  modules : 0xbf000000 - 0xbfe00000   (  14 MB)

    .text : 0xc0008000 - 0xc05cde60   (5912 kB)

    .init : 0xc05ce000 - 0xc0602000   ( 208 kB)

    .data : 0xc0602000 - 0xc0638728   ( 218 kB)

     .bss : 0xc0638728 - 0xc06a8af4   ( 449 kB)
```

```
/**
 *   __vmalloc_node_range  -  allocate virtually contiguous memory
 *   @size:          allocation size
 *   @align:         desired alignment
 *   @start:         vm area range start
 *   @end:           vm area range end
 *   @gfp_mask:  flags for the page level allocator
 *   @prot:          protection mask for the allocated pages
 *   @node:          node to use for allocation or NUMA_NO_NODE
 *   @caller:   caller's return address
 *
 *   Allocate enough pages to cover @size from the page level
 *   allocator with @gfp_mask flags.  Map them into contiguous
 *   kernel virtual space, using a pagetable protection of @prot.
 */

void *__vmalloc_node_range(unsigned long size, unsigned long align,
                        unsigned long start, unsigned long end, gfp_t gfp_mask,
                        pgprot_t prot, int node, const void *caller);


static void *__vmalloc_node(unsigned long size, unsigned long align,
                            gfp_t gfp_mask, pgprot_t prot,
                            int node, const void *caller)
{
        return __vmalloc_node_range(size, 1, 0xbf000000, 0xbfe00000,
                                GFP_KERNEL | __GFP_HIGHMEM,
    PAGE_KERNEL_EXEC,
    NUMA_NO_NODE,
    __builtin_return_address(0));
```

最终在alloc_vmap_area() / vmalloc.c中在[0xbf000000, 0xbfe00000)的virtual address space中分配一块size大小的

空间（virtual address is continuous, but physical address maybe are discrete.）