```
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)
```

在kernel初始化阶段

start_kernel() --> setup_arch() --> paging_init() --> devicemaps_init()

```
static void __init devicemaps_init(const struct machine_desc *mdesc)
{
        struct map_desc map;
        unsigned long addr;
        void *vectors;

        /*
         * Allocate the vector page early.
         */
        vectors = early_alloc(PAGE_SIZE * 2);

        early_trap_init(vectors);

        for (addr = VMALLOC_START; addr; addr += PMD_SIZE)
                pmd_clear(pmd_off_k(addr));


        ......

        /*
         * Create a mapping for the machine vectors at the high-vectors
         * location (0xffff0000).  If we aren't using high-vectors, also
         * create a mapping at the low-vectors virtual address.
         */
        map.pfn = __phys_to_pfn(virt_to_phys(vectors));
        map.virtual = 0xffff0000;
        map.length = PAGE_SIZE;
#ifdef CONFIG_KUSER_HELPERS
        map.type = MT_HIGH_VECTORS;
#else
        map.type = MT_LOW_VECTORS;
#endif
        create_mapping(&map);

        if (!vectors_high()) {
                map.virtual = 0;
                map.length = PAGE_SIZE * 2;
                map.type = MT_LOW_VECTORS;
                create_mapping(&map);
        }

        /* Now create a kernel read-only mapping */
        map.pfn += 1;   devicemaps_init(mdesc);

        map.virtual = 0xffff0000 + PAGE_SIZE;
        map.length = PAGE_SIZE;
        map.type = MT_LOW_VECTORS;
        create_mapping(&map);

        ......
}
```

vectors = early_alloc(PAGE_SIZE * 2);

申请2 pages,从memory的bottom处分配。

early_trap_init(vectors);

```
void __init early_trap_init(void *vectors_base)
{
#ifndef CONFIG_CPU_V7M
        unsigned long vectors = (unsigned long)vectors_base;
        extern char __stubs_start[], __stubs_end[];
        extern char __vectors_start[], __vectors_end[];
        unsigned i;

        vectors_page = vectors_base;

        /*
         * Poison the vectors page with an undefined instruction.  This
         * instruction is chosen to be undefined for both ARM and Thumb
         * ISAs.  The Thumb version is an undefined instruction with a
         * branch back to the undefined instruction.
         */
        for (i = 0; i < PAGE_SIZE / sizeof(u32); i++)
                ((u32 *)vectors_base)[i] = 0xe7fddef1;

        /*
         * Copy the vectors, stubs and kuser helpers (in entry-armv.S)
         * into the vector page, mapped at 0xffff0000, and ensure these
         * are visible to the instruction stream.
         */
        memcpy((void *)vectors, __vectors_start, __vectors_end - __vectors_start);
        memcpy((void *)vectors + 0x1000, __stubs_start, __stubs_end - __stubs_start);

        kuser_init(vectors_base);

        flush_icache_range(vectors, vectors + PAGE_SIZE * 2);
```

```
		modify_domain(DOMAIN_USER, DOMAIN_CLIENT);
#else /* ifndef CONFIG_CPU_V7M */
		/*
		 * on V7-M there is no need to copy the vector table to a dedicated
		 * memory area. The address is configurable and so a table in the kernel
		 * image can be used.
		 */
#endif
}
```

```
	extern char __stubs_start[], __stubs_end[];

	extern char __vectors_start[], __vectors_end[];
```

定义在vmlinux.lds中

```
/*
	* The vectors and stubs are relocatable code, and the
	* only thing that matters is their relative offsets
	*/
__vectors_start = .;
.vectors 0 : AT(__vectors_start) {
 *(.vectors)
}
. = __vectors_start + SIZEOF(.vectors);
__vectors_end = .;
__stubs_start = .;
.stubs 0x1000 : AT(__stubs_start) {
  *(.stubs)
 }
 . = __stubs_start + SIZEOF(.stubs);
 __stubs_end = .;
```

__vectors_start， __vectors_end

__stubs_start， __stubs_end

所代表的其实就是arch/arm/kernel/entry-armv.S中的最原始的trap handler。

整个entry-armv.S中的code分为两部分，分别放在2 sections，以便这两个section被mapping to 2 pages.

early_trap_init()的作用就是把entry-armv.S中的code copy到上面申请的2 pages中。

```
        /*
         * Create a mapping for the machine vectors at the high-vectors
         * location (0xffff0000).  If we aren't using high-vectors, also
         * create a mapping at the low-vectors virtual address.
         */
        map.pfn = __phys_to_pfn(virt_to_phys(vectors));
        map.virtual = 0xffff0000;
        map.length = PAGE_SIZE;
#ifdef CONFIG_KUSER_HELPERS
        map.type = MT_HIGH_VECTORS;
#else
```

```
        map.type = MT_LOW_VECTORS;
#endif
        create_mapping(&map);
```

接着就是把"vector" pages mapping to 0xffff0000。

```
        if (!vectors_high()) {
                map.virtual = 0;
                map.length = PAGE_SIZE * 2;
                map.type = MT_LOW_VECTORS;
                create_mapping(&map);
        }
```

如果ARM CPU并没有把vector page mapping to 0xffff,0000，则mapping to zero page。

```
#define vectors_high()    (get_cr() & CR_V)
#define CR_V (1 << 13) /* Vectors relocated to 0xffff0000    */
```

```
        /* Now create a kernel read-only mapping */
        map.pfn += 1;
        map.virtual = 0xffff0000 + PAGE_SIZE;
        map.length = PAGE_SIZE;
        map.type = MT_LOW_VECTORS;
        create_mapping(&map);
```

接着mapping "stub" page。