页表的映射过程

Arm32

Start\_kernel->setup->arch->paging\_init->map\_lowmem->create\_mapping->\_\_create\_mapping->[alloc\_init\_pud](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pud)->[alloc\_init\_pud](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pud)->[alloc\_init\_pmd](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pmd)->[alloc\_init\_pte](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pte)

Arm64

Start\_kernel->setup->arch->paging\_init->map\_lowmem->create\_mapping->\_\_create\_mapping->[alloc\_init\_pud](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pud)->[alloc\_init\_pud](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pud)->[alloc\_init\_cond\_pmd](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pmd)->init\_pmd->[alloc\_init\_cond\_pte](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pte)

涉及分配代码基本相似，选取一段做描述

mm参数包含了页表的基地址,md描述了一个内存区间，包括虚拟地址的起始地址，物理地址开始地址的页帧号，内存区间大小，内存区间的属性。

alloc参数指定分配函数， ng参数指定pte位拓展支持。

struct **[map\_desc](https://elixir.bootlin.com/linux/latest/C/ident/map_desc)** {

unsigned long **[virtual](https://elixir.bootlin.com/linux/latest/C/ident/virtual)**;

unsigned long **[pfn](https://elixir.bootlin.com/linux/latest/C/ident/pfn)**;

unsigned long length;

unsigned int type;};

static void **[\_\_init](https://elixir.bootlin.com/linux/latest/C/ident/__init)** **[\_\_create\_mapping](https://elixir.bootlin.com/linux/latest/C/ident/__create_mapping)**(struct **[mm\_struct](https://elixir.bootlin.com/linux/latest/C/ident/mm_struct)** \***[mm](https://elixir.bootlin.com/linux/latest/C/ident/mm)**, struct **[map\_desc](https://elixir.bootlin.com/linux/latest/C/ident/map_desc)** \***[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**,

void \*(\***[alloc](https://elixir.bootlin.com/linux/latest/C/ident/alloc)**)(unsigned long **[sz](https://elixir.bootlin.com/linux/latest/C/ident/sz)**),

**[bool](https://elixir.bootlin.com/linux/latest/C/ident/bool)** **[ng](https://elixir.bootlin.com/linux/latest/C/ident/ng)**){

unsigned long addr, length, end;

**[phys\_addr\_t](https://elixir.bootlin.com/linux/latest/C/ident/phys_addr_t)** **[phys](https://elixir.bootlin.com/linux/latest/C/ident/phys)**;

const struct **[mem\_type](https://elixir.bootlin.com/linux/latest/C/ident/mem_type)** \*type;

**[pgd\_t](https://elixir.bootlin.com/linux/latest/C/ident/pgd_t)** \***[pgd](https://elixir.bootlin.com/linux/latest/C/ident/pgd)**;

type = &**[mem\_types](https://elixir.bootlin.com/linux/latest/C/ident/mem_types)**[**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->type]; //取type

#ifndef **[CONFIG\_ARM\_LPAE](https://elixir.bootlin.com/linux/latest/K/ident/CONFIG_ARM_LPAE)**

*/\* \* Catch 36-bit addresses \*/*

if (**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->**[pfn](https://elixir.bootlin.com/linux/latest/C/ident/pfn)** >= 0x100000) {

**[create\_36bit\_mapping](https://elixir.bootlin.com/linux/latest/C/ident/create_36bit_mapping)**(**[mm](https://elixir.bootlin.com/linux/latest/C/ident/mm)**, **[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**, type, **[ng](https://elixir.bootlin.com/linux/latest/C/ident/ng)**);

return;

}#endif

addr = **[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->**[virtual](https://elixir.bootlin.com/linux/latest/C/ident/virtual)** & **[PAGE\_MASK](https://elixir.bootlin.com/linux/latest/C/ident/PAGE_MASK)**; //虚拟地址起始地址 低12位清0（4K页面）

**[phys](https://elixir.bootlin.com/linux/latest/C/ident/phys)** = **[\_\_pfn\_to\_phys](https://elixir.bootlin.com/linux/latest/C/ident/__pfn_to_phys)**(**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->**[pfn](https://elixir.bootlin.com/linux/latest/C/ident/pfn)**);//根据页帧号 获取物理地址

length = **[PAGE\_ALIGN](https://elixir.bootlin.com/linux/latest/C/ident/PAGE_ALIGN)**(**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->length + (**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->**[virtual](https://elixir.bootlin.com/linux/latest/C/ident/virtual)** & ~**[PAGE\_MASK](https://elixir.bootlin.com/linux/latest/C/ident/PAGE_MASK)**));//内存长度+虚拟起始地址低12位后，清零低12位

if (type->**[prot\_l1](https://elixir.bootlin.com/linux/latest/C/ident/prot_l1)** == 0 && ((addr | **[phys](https://elixir.bootlin.com/linux/latest/C/ident/phys)** | length) & ~**[SECTION\_MASK](https://elixir.bootlin.com/linux/latest/C/ident/SECTION_MASK)**)) {

**[pr\_warn](https://elixir.bootlin.com/linux/latest/C/ident/pr_warn)**("BUG: map for 0x%08llx at 0x%08lx can not be mapped using pages, ignoring.\n",

(long long)**[\_\_pfn\_to\_phys](https://elixir.bootlin.com/linux/latest/C/ident/__pfn_to_phys)**(**[md](https://elixir.bootlin.com/linux/latest/C/ident/md)**->**[pfn](https://elixir.bootlin.com/linux/latest/C/ident/pfn)**), addr);

return;

}

**[pgd](https://elixir.bootlin.com/linux/latest/C/ident/pgd)** = **[pgd\_offset](https://elixir.bootlin.com/linux/latest/C/ident/pgd_offset)**(**[mm](https://elixir.bootlin.com/linux/latest/C/ident/mm)**, addr);//根据addr，以及mm->pgd获取addr对应的pgd index

end = addr + length;//确定结束的虚拟地址

do {

unsigned long next = **[pgd\_addr\_end](https://elixir.bootlin.com/linux/latest/C/ident/pgd_addr_end)**(addr, end);//返回下一个pgd的开始位置或者end结束

**[alloc\_init\_pud](https://elixir.bootlin.com/linux/latest/C/ident/alloc_init_pud)**(**[pgd](https://elixir.bootlin.com/linux/latest/C/ident/pgd)**, addr, next, **[phys](https://elixir.bootlin.com/linux/latest/C/ident/phys)**, type, **[alloc](https://elixir.bootlin.com/linux/latest/C/ident/alloc)**, **[ng](https://elixir.bootlin.com/linux/latest/C/ident/ng)**);//申请当前pgd对应的下一级pud空间

**[phys](https://elixir.bootlin.com/linux/latest/C/ident/phys)** += next - addr;//物理地址+PUD\_SIZE

addr = next;//下一个pud的开始位置

} while (**[pgd](https://elixir.bootlin.com/linux/latest/C/ident/pgd)**++, addr != end);}//循环，直到下一个addr==end