arm页表的组织结构

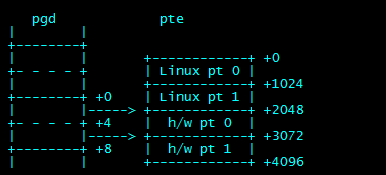
mmu short descriptor format

标准的arm二级页表是4096个一级entry，256个二级entry。

arch/arm/include/asm/pgtable-2level.h

<https://stackoverflow.com/questions/38343748/arm-linux-page-tables-layout>

12-8-12



怎么理解这张图，原来是4096对256，现在也是4096对256，不过是2048\*(2\*4)来组织，256依然也是256，只不过是128\*(2\*4),并且前面有对应的linux pt0，凑成4096利于管理。

linux pt0是用来做模拟功能用，how to emulate。

arm没有dirty young 和access的bit，但是MMU代码需要，所以需要linux pt取模拟之。

when handling page fault in handle\_pte\_fault，pte\_file,pte\_mkdirty,pte\_young,get invoke with the version H/W PTE.

the dirty bit is emulated by only granting hardware write permission if the page is marked ‘writable’ and ‘dirty’ in the linue pte. this means that a write to a clean page will cause a perssion fault,

<https://stackoverflow.com/questions/32943129/how-does-arm-linux-emulate-the-dirty-accessed-and-file-bits-of-a-pte/>

<https://www.coder.work/article/6200855>

At the hardware level, ARM supports two page table trees simultaneously, using the hardware registers TTBR0 and TTBR1. A virtual address is mapped to a physical address by the CPU depending on settings in TTBRC. This control register has a field which sets a split point in the address space. Addresses below the cutoff value are mapped through the page tables pointed to by TTBR0, and addresses above the cutoff value are mapped through TTBR1. TTBR0 is unique per-process, and is in current->mm.pgd (That is, current->mm.pgd == TTBR0 for that process). That is, when a context switch occurs, the kernel sets TTBR0 to the current->mm.pgd for new process. TTBR1 is global for the whole system, and represents the page tables for the kernel. It is referenced in the global kernel variable swapper\_pg\_dir. Note that both of these addresses are virtual addresses. You can find the physical address of the first-level page table by using virt\_to\_phys() functions on these addresses.