Homework1

1. On a 32-bit machine, the huge page size is 4M. On a 64-bit machine, the size of a huge page is 2M. Why not the same size?

在32位机器上，虚拟地址被分成3部分：

位	31-22	21-12	11-0
用途	0级页表索引	1级页表索引	页内索引

大页大小：2^22 = 4MB

在64位机器上，虚拟地址被分成6部分：

位	63-48	47-39	38-30	29-21	20-12	11-0
用途	全0	0级页表索引	1级页表索引	2级页表索引	3级页表索引	页内索引

大页大小：2^21 = 2MB

2. Why OS/MMU use multi-level page tables to map the virtual address to physical address? What is the maximum spatial overhead for the 4-level page table?

如果使用单级页表的话，对于64位机器来说页表大小会有33554432GB。多级页表允许页表中出现空洞，所以可以省去部分未进行映射的页表，可以减少空间占用。

最大空间开销（每一项8B）：

(2^9 + 2^9 * 2^9 + 2^9 * 2^9 * 2^9 + 2^9 * 2^9 * 2^9 * 2^9) * 8 = 550831656960B = 550GB

3. In ARM-smmu architecture, how can mmu distinguish whether it is a page entry or invalid entry?

第3级页表中的页表项中，第0位表示该项是否有效。

4. Please give the advantages and the disadvantages of using block entry / huge page, and give one scenario for each case.

使用大页的优点是可以减少内存访问过程中查找页表的次数，提高性能，同时减少页表占用的空间；缺点是让操作系统控制的最小内存块变大，可能造成一定程度的浪费。使用大页时，减少swap次数，降低访问内存开销，降低TLB不命中的概率，可以提升Oracle的运行性能效率。而当应用数量比较多时，大页的使用会导致每个应用都拥有比较多的内存浪费掉，加剧缺页异常的产生。

5. Memory attribution bit AP and UXN in page entry can already isolate the kernel space and user space, so why ARM-smmu architecture still needs two ttbr registers (Translation Table Base Register), please give a

scenario that two ttbr registers can protect the os but attribution-based isolation can not.

Meltdown攻击。如果内核和用户页表放在一起，仅靠属性判断权限，会导致权限判断延迟，从而在CPU分支预测过程中得以访问无权限的数据；而两个页表分开的话，在用户态下无法查到内核的数据在哪里，也就避免了上述情况。

6. TLB (Translation Lookaside Buffer) can cache the translation from a virtual address to a physical address. However, TLB will be flushed after a context switch between processes. Why? Is it necessary to flush TLB when switching between a user application and the OS kernel? Why?

每一个进程的虚拟地址空间是相同，但是它们对应的物理地址不一样，所以在上下文切换以后，TLB中的映射关系对于当前进程来说是错误的。没有必要，因为用户态和内核态所在的虚拟地址空间不同，所以TLB中的数据虽然是无效的，但不会导致内核的虚拟地址映射到错误的地方。

7. Before ARMv8 architecture, there is no DBM(Dirty Bit Modifier) bit in memory attribution. That’s means hardware does not support dirty page. So how can os simulate this procedure and record the dirty page? Please give a possible solution.

建立页表时，将该页的控制读写权限的一位置为只读，而在软件中将其标记为可读可写，当有对这个页的写操作时，硬件会触发异常，而内核会检查异常的原因，发现该页本身是可写的，这意味着该页变成了dirty page，内核做相应记录后，将读写权限位置为可读可写，然后就可以继续执行了，这时内核已经知道了该页被写过了。