1.练习1：理解基于FIFO的页面替换算法（思考题）

描述FIFO页面置换算法下，一个页面从被换入到被换出的过程中，会经过代码里哪些函数/宏的处理（或者说，需要调用哪些函数/宏），并用简单的一两句话描述每个函数在过程中做了什么？（为了方便同学们完成练习，所以实际上我们的项目代码和实验指导的还是略有不同，例如我们将FIFO页面置换算法头文件的大部分代码放在了kern/mm/swap\_fifo.c文件中，这点请同学们注意）

至少正确指出10个不同的函数分别做了什么？如果少于10个将酌情给分。我们认为只要函数原型不同，就算两个不同的函数。要求指出对执行过程有实际影响,删去后会导致输出结果不同的函数（例如assert）而不是cprintf这样的函数。如果你选择的函数不能完整地体现”从换入到换出“的过程，比如10个函数都是页面换入的时候调用的，或者解释功能的时候只解释了这10个函数在页面换入时的功能，那么也会扣除一定的分数。

·1.do\_pgfault：负责处理缺页异常，通过查找 VMA 确定地址有效性，分配新的物理页或从磁盘加载页面，建立逻辑地址到物理地址的映射，并更新页面的置换状态。

·2.assert：验证条件是否为真，如果为假会终止程序运行并报告错误。

·3.find\_vma：在当前进程的虚拟内存区域列表中查找给定地址所在的 VMA。如果找到了，返回该 VMA；否则返回 NULL。它还使用缓存机制来加速查找操作。

·4.get\_pte：用于获取或创建一个虚拟地址对应的页表项，如果对应的页表项不存在，则根据需要为该虚拟地址分配新的物理内存页，并更新页目录。

·5.pgdir\_alloc\_page：为指定的逻辑地址（la）分配一个物理页面，并将该页面插入到页目录中。

·6.swap\_out：负责将内存中的页面交换到磁盘，释放内存空间，为其他页面腾出资源。

swap\_in：负责将磁盘上的页面加载回内存，恢复之前交换出去的页面，保证页面可以继续使用。

·7.swapfs\_read：将指定 swap\_entry\_t 对应的交换空间页从磁盘设备读取到物理内存中指定的页面page中。

·8.page\_insert：将页面 page 映射到线性地址 la，并根据需要更新或替换已有的页表项，同时维护页面引用计数和刷新 TLB。

·9.swap\_map\_swappable：通过调用当前页面替换算法的 map\_swappable 方法，将页面 page 标记为可替换，并与地址 addr 关联，确保该页面能够被替换算法正确管理。

·10.\_fifo\_swap\_out\_victim：实现了从 FIFO 队列中选择一个受害页面的逻辑，移除该页面并返回其地址，用于页面替换操作。在整个页面替换过程中，它起到了页面选择的作用。

·11.swapfs\_write：将内存页面的数据写入磁盘交换区，释放内存资源。它利用交换条目定位磁盘位置，并通过底层 ide\_write\_secs 实现写入操作。这是实现页面换出的核心步骤。

·12：free\_page：释放页面，完成换出。

·13：tlb\_invalidate：分配一个物理页面，并在页表中建立线性地址到物理地址的映射，同时处理换页相关设置。刷新TLB。

练习2：深入理解不同分页模式的工作原理（思考题）

get\_pte()函数（位于kern/mm/pmm.c）用于在页表中查找或创建页表项，从而实现对指定线性地址对应的物理页的访问和映射操作。这在操作系统中的分页机制下，是实现虚拟内存与物理内存之间映射关系非常重要的内容。

get\_pte()函数中有两段形式类似的代码， 结合sv32，sv39，sv48的异同，解释这两段代码为什么如此相像。

目前get\_pte()函数将页表项的查找和页表项的分配合并在一个函数里，你认为这种写法好吗？有没有必要把两个功能拆开？

·不同的虚拟地址位数：在不同的虚拟内存模型中（如 sv32, sv39, sv48），虚拟地址的位数不同，导致页表的层数和每层的索引位也不同。例如，sv32 使用 32 位虚拟地址，sv39 使用 39 位虚拟地址，sv48 使用 48 位虚拟地址。不同的页表模型可能有 2 层（如 sv32）或 3 层（如 sv39 和 sv48）的页表，因此代码中的页表项查找（如 pde\_t \*pdep1 = &pgdir[PDX1(la)]）会根据虚拟地址的不同层级进行调整。

·代码的相似性：这两段代码的相似性主要在于：首先通过索引进入页表的某一层（pde\_t \*pdep1 = &pgdir[PDX1(la)]），然后检查该层的页表项是否有效，如果无效且需要创建（create 参数为真），就分配一个新页面，并将其作为当前层的页表项。接着，通过映射关系找到目标页表项并返回。

·不同层级的页表：不同虚拟内存模型的页表索引部分可能略有不同。例如，sv32 会使用较低的 10 位作为页目录项索引，而 sv39 和 sv48 则使用更多的位来索引页表层级，这些细节会体现在 PDX1 和类似的宏定义中。

2.

优点：

·简化代码逻辑：将查找和分配功能合并可以减少调用的复杂度，尤其是在某些需要创建页表项的场景中，一步到位的做法更为高效。

·提高代码的可读性：在实现某些页表管理操作时，通常会同时检查页表项是否存在并进行创建，这种一体化处理在某些情况下是比较直观的。

缺点：

·不利于单一职责原则：将两种功能（查找和创建）混合在一个函数中，违反了单一职责原则。理想情况下，一个函数应该只做一件事，要么查找，要么创建，便于维护和扩展。

·潜在的错误和不易调试：如果这两个功能过于耦合，可能会导致调试时不容易分辨问题的根源。如果一个功能发生错误，可能需要对两者进行全面检查。

拆分好处：将这两个功能拆分成单独的函数，如 get\_pte() 专门负责查找页表项，create\_pte() 或 alloc\_pte() 专门负责创建页表项。这样做可以更清晰地管理内存和页表，便于后续扩展和调试。其次，从性能的角度来看，合并两者可能会更高效，因为每次查找时都可能需要创建新页表项，而将两者分开则可能带来不必要的多次函数调用。然而，从可维护性和灵活性角度出发，拆开会更好。