问题 1

（1）编译阶段：buddy.h 指定了 ChCore 物理内存布局。该文件定义了 struct page，即页面元数据的结构；还定义了 struct phys_mem_pool，即物理内存的结构。

（2）运行时阶段：mm.c 指定了 ChCore 物理内存布局。该文件对页面元数据和 free lists 进行了初始化。

练习 1

我采用以下顺序完成这个练习：

split_page()函数：该函数负责把一个较大 order 的内存块分成两个相等大小的伙伴内存块，它们的 order 都是原 order 减 1。

buddy_get_pages()函数：该函数负责寻找一个指定 order 的内存块，当 free_lists 里面存在指定 order 的元素时，则直接分配；不存在时则要使用 split_page()函数对较大 order 的内存块进行分解。

merge_page()函数：该函数负责将指定内存块与其伙伴归并成一个大内存块，通过递归持续归并直到不能再归并为止。

buddy_free_pages()函数：该函数负责将一个已经分配的块释放掉，这里要用到 merge_page()函数将该块与伙伴归并。

问题 2

AArch64 采用了两个页表基地址寄存器，TTBR1_EL1 寄存器用于存储内核映射的页表的基地址，TTBR0_EL1 寄存器用于存储用户态程序的映射的页表的基地址。性能方面，由于内核和应用程序使用不同的页表，所以在进行系统调用时不需要切换页表，大大提高了系统调用的性能。安全方面，用户态和内核态页表基地址寄存器的分离可以保护内核态页表不被用户态读到，保护了内核的安全。

问题 3

1. 物理地址。
2. 物理地址

问题 4

1. 需要 1 个 L0 页表，1 个 L1 页表，4 个 L2 页表，2^{11} 个 L3 页表。所以页表大小为$(1 + 1 + 4 + 2^{11}) * 4K \approx 8M$，即管理内存需要 8M 空间。
2. （1）AArch64 采用了两个页表基地址寄存器 TTBR0_EL1 和 TTBR1_EL1，而 x86_64 只有一个页表基地址寄存器 CR3。（2）AARCH64 上内核和应用程序使用不同的页表，而 x86_64 上内核和应用程序使用相同的页表，内核映射到应用页表的高地址。（3）x86_64 支持分段和分页，AARCH64 只支持分页。

AArch64 MMU 架构设计较 x86_64 来说提高了性能，且具有更好的隔离性。

练习 2

query_in_pgtbl()函数：该函数的作用是将虚拟地址翻译成物理地址，函数的第一个参数是一级页表的基地址，通过这个基地址，虚拟地址可以很容易地翻译成物理地址。

map_range_in_pgtbl()函数：该函数的作用是将指定大小的空间的虚拟地址与物理地址进行映射，本质上就是设置 L3 页表存的值。

unmap_range_in_pgtbl()函数：该函数的作用是使指定大小的空间的虚拟地址失效，方法是设置页表的相应 is_valid 字段为 0。

问题 5

仍然需要。用户态程序仍然可以通过访问高地址来对内核态地址进行操作，所以为了隔离内核态和用户态的地址空间，必须要设置页表位的属性。

问题 6

（1）内核内存很大，一个 4K 的 page 相对来说很小，所以 ChCore 采用了更粗粒度的块条目（2MB）来组织内核内存。
内核空间在 Boot 阶段必须映射，因为它要被用来启动内核。其他空间可以在内核启动后延迟。
（2）用户代码被认为是不可靠的，它可能会造成崩溃以及安全隐患，如用户数据覆盖内核数据、恶意的用户态进程修改内核数据等。

为了保护内核内存，使用专门的 TTBR1_EL1 寄存器用于存储内核映射的页表的基地址，并且通过设置页表位的属性来隔离内核态和用户态的地址空间。

练习 3

map_kernel_space()函数与 map_range_in_pgtbl()函数相似，只是它的作用是将指定大小的空间内核态虚拟地址与物理地址进行映射。属性位文档已经给出，照着写便是。