CS3601-Lab2

陈景韬

523030910028

Q1

练习题1-完成 kernel/mm/buddy.c 中的 split_chunk、merge_chunk、buddy_get_pages、 和 buddy_free_pages 函数中的 LAB 2 TODO 1 部分，其中 buddy_get_pages 用于分配指定阶大小的连续物理页，buddy_free_pages 用于释放已分配的连续物理页。

split_chunk和merge_chunk都可以用递归的方法进行实现，注意merge_chunk的时候块的order不能超过MAX_ORDER

buddy_get_pages循环迭代找到free_list中能容纳的最小规模，且要求未被allocated，然后再split到合适的大小。

buddy_free_pages则是释放指定位置的块，后面再调用merge将可能的空闲块合并即可。

注意加上对空页的判别，不知道scores环节会不会特意传一个空的page。

Q2

练习题2-完成 kernel/mm/slab.c 中的 choose_new_current_slab、alloc_in_slab_impl 和 free_in_slab 函数中的 LAB 2 TODO 2 部分，其中 alloc_in_slab_impl 用于在 slab 分配器中分配指定阶大小的内存，而 free_in_slab 则用于释放上述已分配的内存。

choose_new_current_slab 从pool中的partial_slab选一个slab来用，如果没有就返回NULL

alloc_in_slab_impl 从slab中找一个slot，如果满了就调用choose_new_current_slab再找一个新的slab

free_in_slab 将slab添加回free_list

Q3

练习题3-s完成 kernel/mm/kmalloc.c 中的 _kmalloc 函数中的 LAB 2 TODO 3 部分，在适当位置调用对应的函数，实现 kmalloc 功能

size小的时候使用slab来分配，调用alloc_in_slab

size大的时候使用buddy_system来分配，调用get_pages()

Q4

练习题4-完成 kernel/arch/aarch64/mm/page_table.c 中的 query_in_pgtbl、

map_range_in_pgtbl_common、unmap_range_in_pgtbl 和 mprotect_in_pgtbl 函数中的 LAB 2 TODO 4 部分，分别实现页表查询、映射、取消映射和修改页表权限的操作，以 4KB 页为粒度。

query_in_pgtbl 逐级遍历列表，如果找不到条目就返回错误；中途额外处理大页的情况，直接返回所需的物理内存映射即可。如果正常找到最后一级页表，同样计算物理地址并返回entry。

map_range_in_pgtbl_common 批量映射一批连续的虚拟地址区间到物理内存空间，直接进到L3表，然后遍历空间并新建对应的条目，填写相关位数。

unmap_range_in_pgtbl 类似query_in_pgtbl中逐级遍历列表，但是注意当返回-ENOMAPPING的时候，说明此级页表下所有vm都未被映射，因此直接skip这一大段，不用unmap。如果正常到达末级页表，就逐项将其unmap，当前项用完或者total耗尽就停止。

mprotect_in_pgtbl 类似map_range_in_pgtbl直接进到末级表，然后逐项修改权限

Q5

思考题5-阅读 Arm Architecture Reference Manual，思考要在操作系统中支持写时拷贝（Copy-on-Write，CoW）1需要配置页表描述符的哪个/哪些字段，并在发生页错误时如何处理。（在完成第三部分后，你也可以阅读页错误处理的相关代码，观察 ChCore 是如何支持 Cow 的）

要配置AP(Access Permissions)，两位数分别定义el1和el0下的权限。

在发生页错误的时候，首先判断是否为CoW页面。如果是CoW，就分配一个新的物理页，并修改当前进程页表，建立对应vm-pm映射。然后恢复进程执行，完成操作。当然还要把对应结构体的AP改过来，不然下次还会报pagefault。非CoW的话直接报错，终止进程。

Q6

思考题6-为了简单起见，在 ChCore 实验 Lab1 中没有为内核页表使用细粒度的映射，而是直接沿用了启动时的粗粒度页表，请思考这样做有什么问题。

在权限管理方面会出问题，粗粒度的映射无法对小段内存进行精细的权限划分，进而导致安全问题，比如只读区域被映射到一个读写大页，导致内存的隔离安全被严重破坏；

同时也会导致大量的内存浪费，如果只需要映射少量的内容，粗粒度映射依然会为其分配大段内存，导致大量的内部碎片的产生。

Q8

完成 kernel/arch/aarch64/irq/pgfault.c 中的 do_page_fault 函数中的 LAB 2 TODO 5 部分，将缺页异常转发给 handle_trans_fault 函数。

调用handle_trans_fault函数，传递当前进程的虚拟内存空间以及出错的地址，交给函数去补齐映射

Q9

练习题9-完成 kernel/mm/vmspace.c 中的 find_vmr_for_va 函数中的 LAB 2 TODO 6 部分，找到一个虚拟地址找在其虚拟地址空间中的 VMR。

在红黑树上搜索，调用rb_search()，查找addr所属的区域，然后调用rb_entry()来得到所属的指针，返回对应的区间struct的指针。

Q10

练习题10-完成 kernel/mm/pgfault_handler.c 中的 handle_trans_fault 函数中的 LAB 2 TODO 7 部分（函数内共有 3 处填空，不要遗漏），实现 PMO_SHM 和 PMO_ANONYM 的按需物理页分配。你可以阅读代码注释，调用你之前见到过的相关函数来实现功能。

在第一个blank处，调用virt_to_phys()将va转换成pa，然后用memset()把页空间清零。

在第二个blank处调用map_range_in_pgtbl()来建立映射，缺乏的rss参数一般用来进行内存占用统计，这里无关紧要，自定义一个占位用就可以。

在第三个blank处和第二个blank一样的调用map_range_in_pgtbl()，用于防止多进程映射中不同进程的冲突问题，再映射一遍可以防止映射丢失，且本身不会影响到正常进程中的运作。