Lab2 实验报告

任务描述

完成 xv6 的虚存管理

arm 虚拟地址转换

pgdir_walk

依次查找第 0 ~ 3 级页表，通过地址相应位上的偏移量找到的对应的页表描述符，把后 12 位设 0 后进入下一级页表。

map_region

枚举虚拟地址，使用 `pgdir_walk` 函数得到对应的三级页表入口的指针，修改该位置上的值为对应的物理地址。

vm_free

递归执行，枚举当前页表中所有的项，找到合法的页表描述符，继续清空该页表。

更新 TLB

arm 用前16 位全 1 (0xFFFF 开头) 表示使用内核地址空间，此时使用 TTBR1_EL1 作为 0 级页表入口；前 16 位全 0 表示使用用户地址空间，此时使用 TTBR0_EL1 作为 0 级页表入口。当在内存中创建新页表之后，CPU 的 MMU 中仍然没有更新相应的 TLB 。需要使用 `asm volatile` 语句调用汇编指令，手动修改 TTBRn_EL1 ，之后再清空 CPU 的指令、数据缓存，并且刷新 MMU 的所有 TLB 。

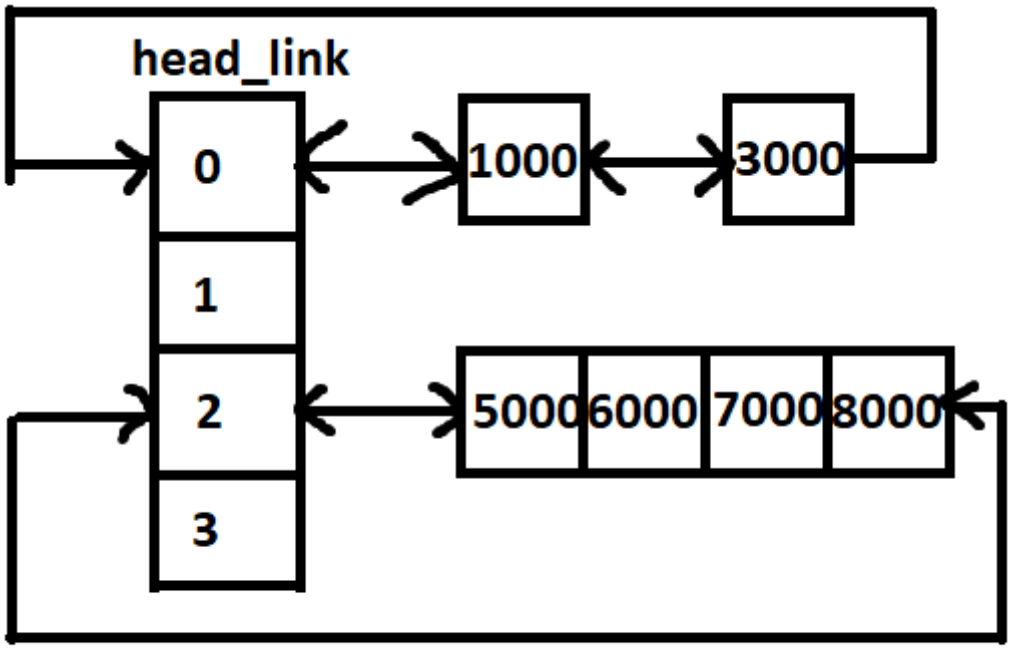
页分配

在 `buddy_kalloc.c` 中使用了 *buddy_system* 作为页分配的算法。

在实现中，先把要分配空间的头部一部分取出，存放该段内存空间的控制器 allocator ，主要管理以下数据。

- status： 存储每个页表开头的连续未分配空间对应长度为 2 的多少次方
- link_head： 存储第一块长度为 2 的 x 次方的连续未分配空间的指针

此外，对每块未分配页，在该页初始位置存储该页对应的连续未分配长度接下去一段未分配空间的开始位置，和对应的 link_head 组成双向链表



status = [0, 1, 0, 1, 3, 0, 0, 0]

alloc

在 head_link 数组中从对应长度开始往大的长度找，找到第一个可以分配的连续空闲页；

之后把当前的连续空闲页分成左右等长的两部分，把右半部分插入到对应长度的链表中，继续对左半部分执行上述操作直到长度刚好为止。

free

把需要释放的内存段插入到对应的链表中；

查看当前长度下相邻的内存段是否全部空闲 (status)，如果空闲，合并更新后处理长度 * 2 时的情况，否则退出。

复杂度为 $log_2(maxsize)$，但需要消耗空间存储 status 。

测试程序

在 `main.c` 中分别对页分配和地址转换各进行了 1 个测试。

页分配：分 2 次分配了 4 页的空间给 a 和 b 。之后把从 a 开始长为 4 个页的空间释放，从 b 开始长为 2 个页的空间释放，从 $b+8192$ 开始长为 2 个页的空间释放。再次请求分配 8 页的空间，此时页分配返回的开始位置应该和 a 相同。

地址转换：先在 0xFFFF_0000_0000_1010 的位置写入数据 233， 即在物理地址 0x0000_0000_0000_1010 的位置写入 233； 之后调用 `map_region` 使 0x0000_0000_0000_0010 映射到 0x0000_0000_0000_1010 ， 更新用户空间的 TLB 之后，访问 0x0000_0000_0000_0010 ， 此时读出的数据应该对应物理地址 0x0000_0000_0000_1010 上的数据 233。