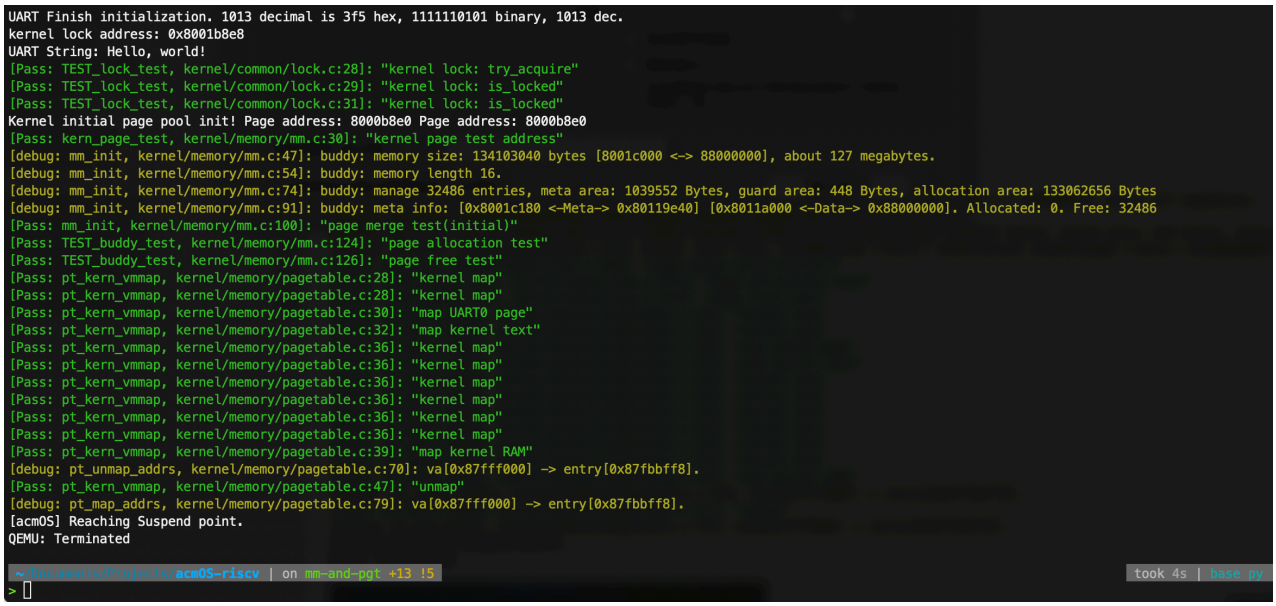
OS Project Part 2: 内存管理

提交须知：本Lab包括至少3个函数填空文件与3个问题，只有输出预期的结果并且回答完问题才可以获得全部分数。

期望结果

你完成本次作业之后，在项目根目录，可以通过`./scripts/docker_build.sh`构建系统镜像。在完成内核的构建之后，可以输入`make qemu-nox`启动qemu并且模拟执行我们构建好的内核。acmOS的标准输出应该显示在QEMU中，如下图。



要退出QEMU，按下Ctrl-A，再输入X退出QEMU。

关于作业

本次需要填写的代码全部为`answer_***.h`，你可以选择使用自己的设计（只要在answer_***.h`的范围内都可以随意更改）。如果你觉得在头文件之外的文件需要修改请联系助教。

本次作业的预期时长为2周，发布2021年5月6日，截止时间2021年5月29日。

预期的代码量为约100行。

请从github上拉取代码，并切换到`mm-and-pgt`分支。

步骤1: 内存管理

这个os的内存包括了两个部分，一个部分的内存通过`static`修饰，（`kernel_page_initialized`），另一部分则是在系统启动之后进行管理。前者可以用于系统部分关键模块的关键内存使用。

需要注意的是，两部分内存的管理方式不相同。在这个demo os中，前者采用一个大数组进行管理，并且每次分配只可以分配1个页。

需要填写的函数：

- `kernel/memory/answer_mm.h:void* kern_page_malloc();`
- `kernel/memory/answer_mm.h:void kern_page_free(void* ptr);`

步骤提示：观察在mm.c文件中`kern_page_init()`与`kern_page_test()`的初始化和测试方式。

除了通过静态数组直接访问的内存之外，剩余部分则是通过伙伴算法进行管理。

如果你有时间和兴趣的话，可以在完成步骤2之后完成以下步骤：

- 修改`kernel/memory/answer_mm.h:void* mm_kalloc();`
  与`kernel/memory/answer_mm.h:void mm_kfree(void* ptr);`，让它变得更简洁。

步骤2: 伙伴算法

在课上已经讲解了伙伴算法，那么现在请你实现一个简单的伙伴算法。这一部分我们管理的是内核静态区和代码段的末尾到`PHYTOP`的内存。助教提供的管理方式如下：

页的元数据区提供一个一一映射到页区域，页的元数据区可以从`kernel/memory/buddy.h`中看到详细信息。

由于伙伴算法并不对外暴露接口，只提供给`mm_kalloc`和`mm_kfree`使用，因此你可以自由设计`answer_buddy.h`的所有内容。你也可以选择采用助教给出的代码完成本次作业。

步骤提示：

1. `_buddy_alloc_page`为分配一个传入秩大小的页，返回页所属的元数据的指针。
2. `buddy_free_page`为给出一个页元数据的指针，释放这个页。

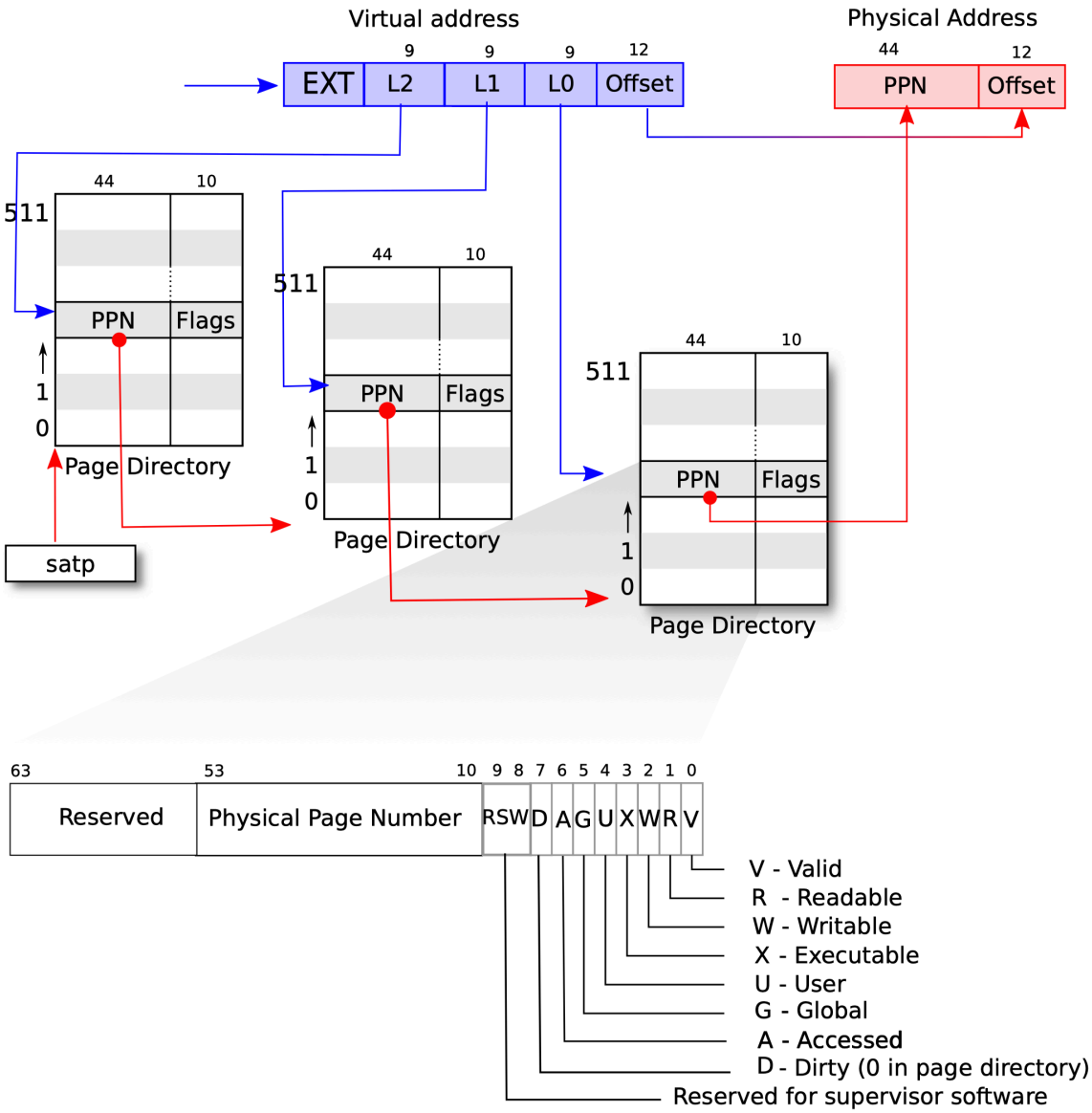
问题：

(1) 可不可以删除页的元数据，直接分配页空间?

(2) 何时应该触发页的合并和页的分裂?

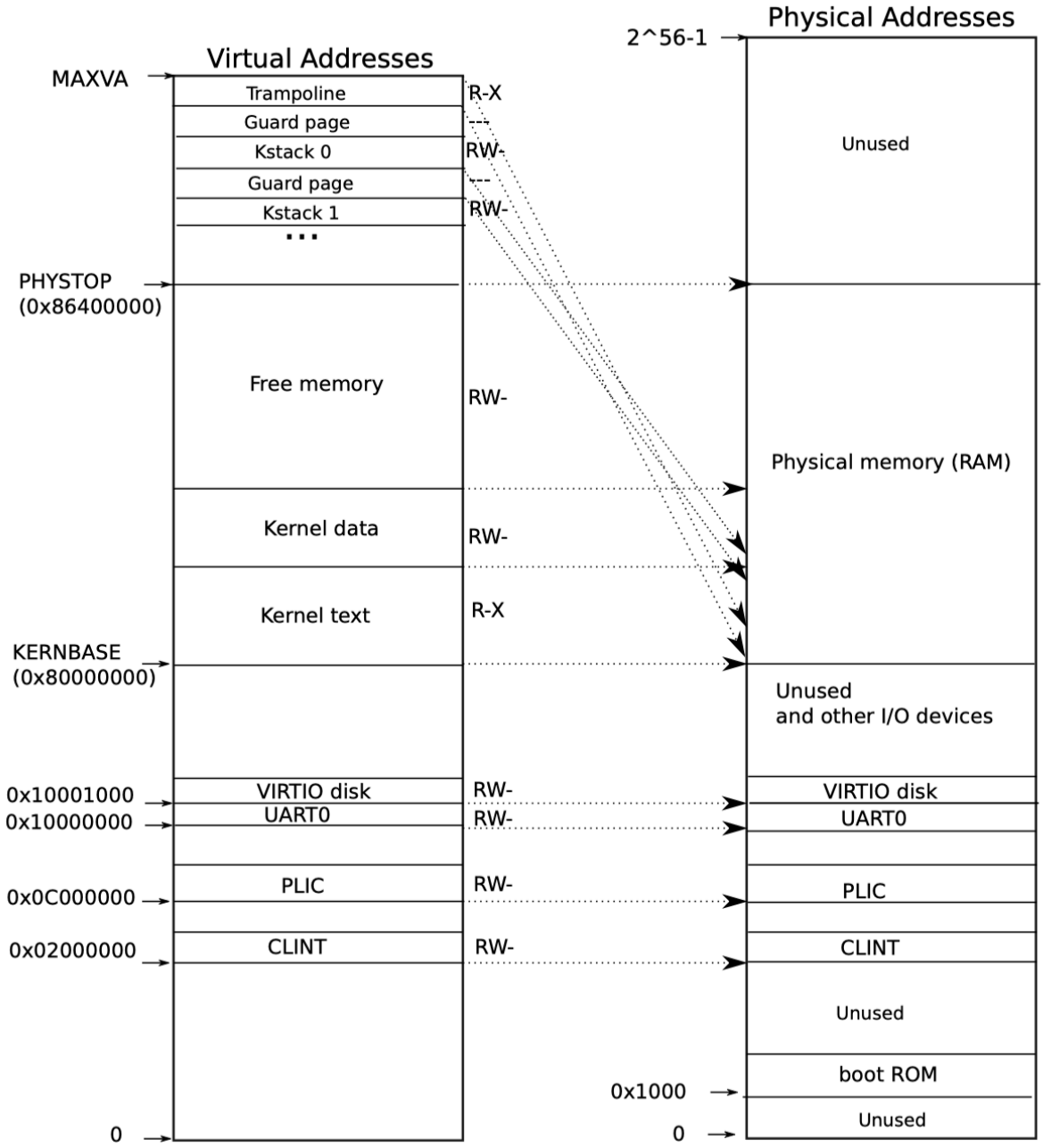
步骤3: 页表管理

有了分配页的算法，我们下一个需要考虑的内存管理相关的话题是页表。以下是RISCV的三级页表模式所对应的图解。



通过SATP找到页的基地址，然后按照页表的方法寻找页。已经给出的代码中采用的都是**4级页表**(SV48)的格式，描述如下：

```
// For VA:
//   47...63 zero
//   39...47 -- 9  bits of level-3 index
//   30...38 -- 9  bits of level-2 index
//   21...29 -- 9  bits of level-1 index
//   12...20 -- 9  bits of level-0 index
//   0...11  -- 12 bits of byte offset within the page
```



上图则描述了这个os的映射关系。为了简化，我们内存并没有设置的非常大，内核部分的映射也已经展现在图上。你需要完成的是启动页表、映射页、解映射三个部分。相关代码位置位于`kernel/memory/answer_pgt.h`。

步骤提示：

1. `static pte_t* pt_query(pagetable_t pagetable, vaddr_t va, int alloc)`为查找页表`pagetable`中`va`的页项，并返回对应页表项的地址，alloc表示在这个页表项不存在的时候是否分配一个页空间。注意，在这里暂时不考虑page fault的问题。
2. `pt_map_pages`为映射`va`到`pa`连续`size`个页，并设置对应的权限。注意，有一项页表项的权限需要**你设置**而不是传入。
3. `pt_map_addrs`和`pt_unmap_addrs`为对单一页表项建立或解除映射。
4. `paddr_t, vaddr_t`实际上都指同一种类型，但是建议不要混合使用防止出现错误。
5. 可以通过循环验证自己的映射是否正确。

问题：

(1) 观察代码和RISCV手册，描述SV39和SV48的异同。

关于评分的补充事项

本次作业采用输出比对+Code Review方式进行。注意你可以修改的文件在没有特殊情况下仅为`answer_***.h`。当然，如果你觉得为了实现这个功能，这样的设计是非常差劲的，你也可以联系`@peterzheng`助教咨询后自主设计。

其余未尽问题，请咨询助教。