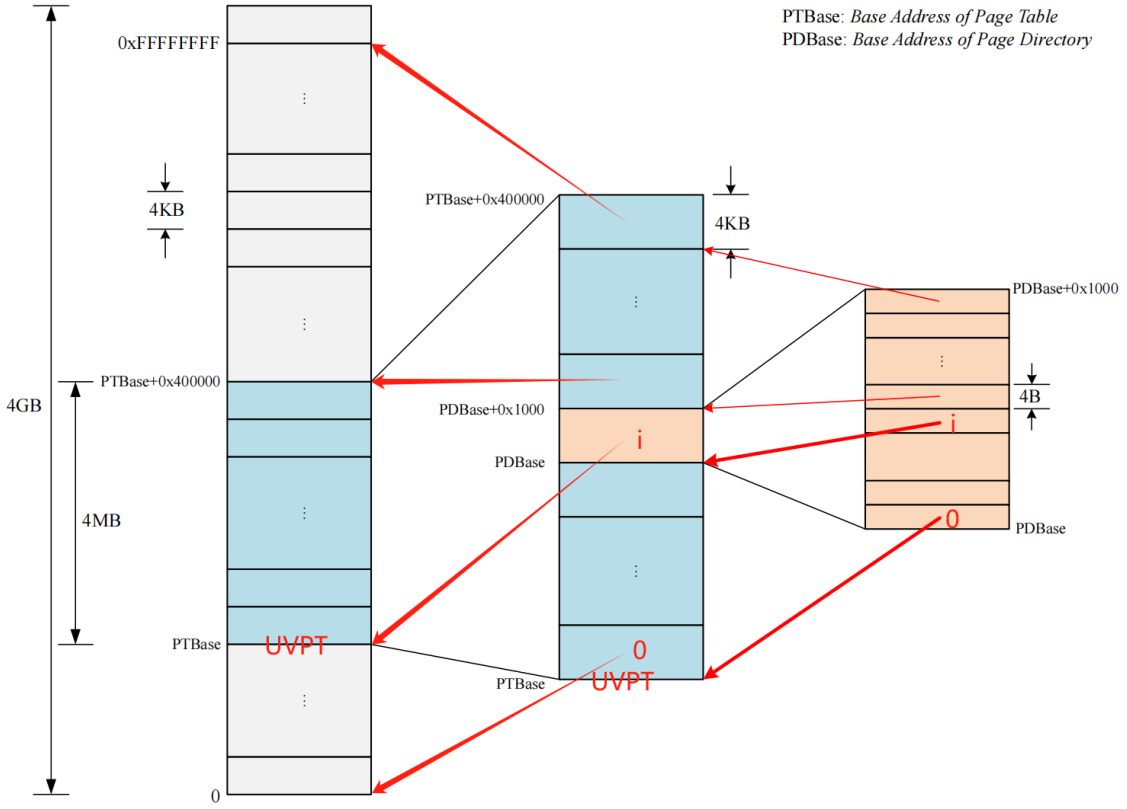
Lab3

> 班级：222111
>
> 学号：22373340
>
> 姓名：詹佳博

思考题

Thinking 3.1

- UVPT为0x7fc00000，也就是图中的PTBase，设第i个页目录项为自映射项(`e->env_pgdir[i]`)。$\frac{UVPT-0}{4M}=\frac{PDBase-UVPT}{4K}=i$，解得i=UVPT>>22，即`PDX(UVPT)`。



- 进程页目录的第`PDX(UVPT)`个**页目录项**保存着该**进程页目录所在页表**的物理地址，也就是形成了自映射。并且给他仅置**可读权限**。

Thinking 3.2

- data是在函数`env_create`通过调用`env_alloc`函数分配的一个新进程。
- 作用是最终传给`load_icode_mapper`，为该进程分配一个新的页目录；如果`map_page`不同，所传入的`data`也有可能不同，从而用`void *`来增强可复用性。
- 不可以。如果没有这个参数，则回调函数map_page会缺少参数无法运行。

Thinking 3.3

- 首先，处理非页面对齐的段起始地址。
  - 如果段的虚拟地址（`va`）不是页面大小的整数倍，则函数首先需要映射包含段起始地址的页面。这个页面的一部分属于该段，而另一部分则不属于。因此第一个页面的映射大小会相应减少，只映射实际占用部分。
- 然后，进行段的文件内容加载。
  - 函数会加载段在文件中对应的所有内容（`bin_size` 大小），通过循环调用 `map_page` 函数实现，每次映射一个页面。每个页面大小均为 `PAGE_SIZE`，直至结束。最后一面根据 `bin_size` 大小分配相应大小页面。
- 最后，用零填充至段的内存大小。
  - 如果段在内存中的大小（`sgsize`）大于其在文件中的大小（`bin_size`），则函数需要分配额外的页面，并用零来填充这些页面。在循环中，`map_page` 函数的 `bin` 参数设置为 `NULL`，表示不需要从文件中加载内容，而是分配新的页面并用零填充。每个页面大小均为 `PAGE_SIZE`，直至结束。最后一面根据 `sgsize` 大小分配相应大小页面。

Thinking 3.4

- 储存的是**虚拟地址**。当CPU试图从这个地址取指令时，MMU会负责将其翻译成对应的物理地址。

Thinking 3.5

- **0 号异常** `handle_int` 的函数实现在 `kern/genex.S`。
- **1 号异常** `handle_mod` 的函数实现在 `kern/genex.S`，其关键函数 `do_tlb_mod(struct Trapframe *tf)` 在 `kern/tlbex.c` 中实现。
- **2 号异常** `handle_tlb` 的函数实现在 `kern/genex.S`，其关键函数 `do_tlb_refill()` 在 `kern/tlb_asm.S` 中实现，其所关联的 `_do_tlb_refill(u_long *pentrylo, u_int va, u_int asid)` 在 `kern/tlbex.c` 中实现。
- **3 号异常** `handle_tlb` 的函数实现同**2 号异常** `handle_tlb` 的函数实现。

Thinking 3.6

- 在调度执行每一个进程之前（env_run），会调用 `env_asm.S` 的 env_pop_tf 函数，进而调用了 include/kclock.h 的 RESET_KCLOCK 和 `genex.S` 的 ret_from_exception 恢复了 Status 寄存器，该进程**时钟中断开启**。
- `genex.S` 的 handle_int 函数根据 Cause 寄存器的值判断是 Timer 对应的 7 号中断位引发的时钟中断，执行中断服务函数 timer_irq，跳转到 schedule 中执行，该进程**时钟中断结束**。

Thinking 3.7

- 在调度执行每一个进程之前（env_run），会调用 `env_asm.S` 的 env_pop_tf 函数，进而调用了 include/kclock.h 的 RESET_KCLOCK 和 `genex.S` 的 ret_from_exception 恢复了 Status 寄存器，**开启了时钟中断**。
- RESET_KCLOCK 重置 CP0 的 Timer，CPU 执行命令，Count寄存器自增，直到与Compare寄存器值相等，时钟中断触发。此时 PC 指向 0x80000180，跳转到 `entry.S` 的 exc_gen_entry。通过它的分发，调用 exception_handlers()，调用 `genex.S` 的 handle_int。

- handle_int 函数根据 Cause 寄存器的值判断是 Timer 对应的 7 号中断位引发的**时钟中断**，执行中断服务函数 timer_irq，跳转到 schedule 中执行，并且 yield == 0（不指定必须发生切换）。若时间片被用完（count == 0），调度开启下一个新进程。该进程**时钟中断结束**。
- 当调度完毕开启下一个进程，继续执行以上流程。

难点分析

1. `Exercise 3.5` 在第二步时，先直接写`memcpy((void *)(offset), src, len);`，后来发现毫无意义，应该是在页面进行位移才有效，同时CPU操作均为虚拟地址，则为`page2kva(p)`。
2. `Thinking 3.3` 处理页面加载的情况有些难想，并且发现在一些情况下，会有一些空间没有被 map_page 函数映射？ 如下图。

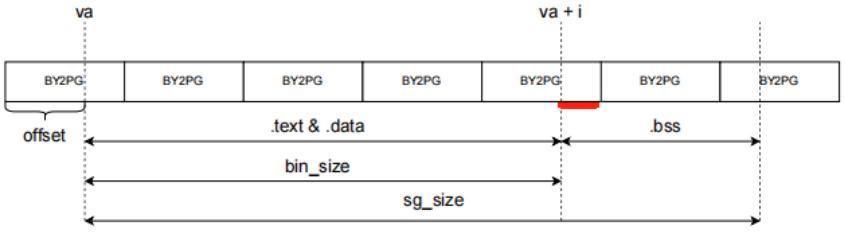


图 3.2: 每个 segment 的加载地址布局

3. `Exercise 3.12` 完成schedule函数的时候，在从 env_sched_list 取出新进程的时候，我同时使用了TAIL_REMOVE 将其移除。经过思考后发现不该移除，而是处于循环内，以防其没有运作完，再下一次循环中应该被置尾端，而不是移除，无法再在链表中访问到。

实验体会

1. 学习了 `try()` 和 `panic_on()` 用法。前者如果函数非0，返回报错 `int` ；后者是一种中断，如果函数非0，直接调用 `panic()` 报错。
2. `Exercise 3.4` 应该是可以在第一步取出e之后，直接进行LIST_REMOVE，可提示将其移到最后一步，应该效果一样。
3. 此次Lab有关env的链表操作与Lab2的Page链表操作十分类似，故而代码写起来比较轻松。Lab3和Lab2在很多地方都可以进行自然而然的类比和迁移，并且使用了很多Lab2的相关函数。
4. 此次上机的exam在我看来不是很难，助教给的提示完全避开了我的错误点（，我错在对count没有累加。。。下次得好好看题；
5. 此次的extra有点难想，但是自己做的快差不多了，可惜时间不够，惨淡收场（