实验报告

PA3-存储管理

班级 大类八班

学号 3022244290

姓名 陈秋澄

实验进度（任务自查表）

序号	完成情况
必做任务 1	已完成
必做任务 2	已完成
必做任务 3	已完成
必做任务 4	已完成
选做任务 1	未完成
选做任务 2	未完成
选做任务 3	未完成

思考题（请注明题号，如思考题 1，思考题 2，...）

思考题 1：GDT 能有多大

你能根据段选择符的结构，计算出 GDT 最大能容纳多少个段描述符吗？

段选择符的结构中，INDEX 有 13 位，故 GDT 最大能容纳 2^13 个段描述符。

思考题 2：为什么是线性地址

GDTR 中存放的 GDT 首地址可以是虚拟地址吗？为什么？

不可以。虚拟地址需要经 GDT 中的段表翻译才能得出地址，而如果 GDTR 中存放虚拟地址则找不到 GDT 在哪里了。

思考题 3：如何提高寻找段描述符的效率?

可以按照高速缓存的思想，建立类似 cache 和 TLB 的结构来提高寻找效率。

思考题 4：段式存储管理的缺点？

分段管理要求分配一大段连续的存储空间，难以实现并且容易造成大量的外部碎片出现。

思考题 5：页式存储管理的优点？

页式存储管理没有外部碎片，并且不再需要大段连续的存储空间，提高了内存的利用率。

思考题 6：一些问题

1、80386 不是一个 32 位的世界吗，为什么表项中的基地址信息只有 20 位，而不是 32 位？

分页基地址有 20 位是 8086 的传统。

在 8086 的分段机制中，每个段的基地址由 seg_reg（即段寄存器的值）<<4 得到，而段寄存器是 16 位的，左移 4 位得到 20 位的基地址。

2、手册上提到表项(包括 CR3)中的基地址都是物理地址，物理地址是必须的吗？能否使用虚拟地址或线性地址？

这是必须的，如果 cr3 中的基地址是虚拟地址，则无从寻找页表翻译成物理地址，进入死循环。至于其他表项的虚拟地址与线性地址问题同理。

3、为什么不采用一级页表？或者说采用一级页表会有什么缺点？

多级页表可以有效地节约内存空间，如果仅采用一级页表，将可能导致较大的页表长期驻留在内存中。

思考题 7：空指针真的是“空”吗？

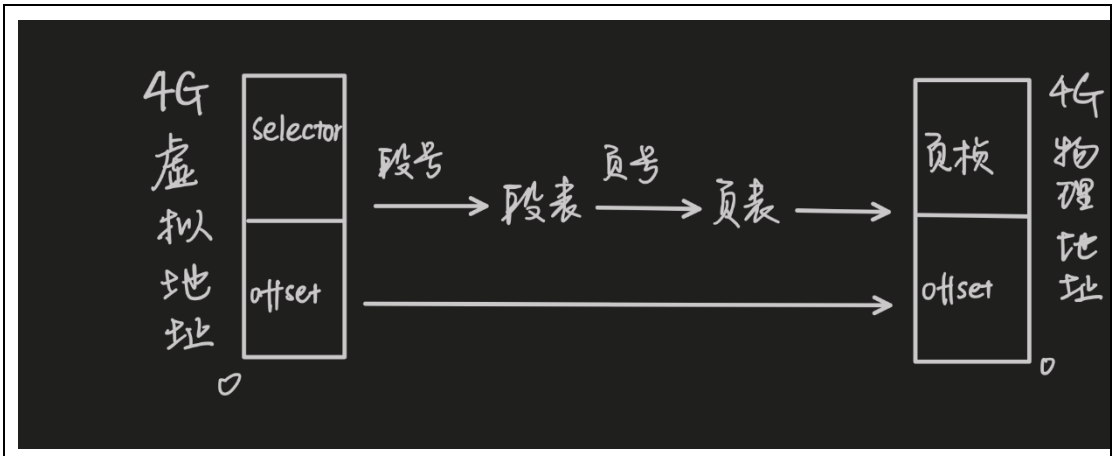
空指针只是未分配或者未指向内存任何位置的指针，并不是“NULL”的。

思考题 8：在扁平模式下如何进行保护？

对于数据有不同的访问权限，未达到需要的权限时不能进行写操作。

思考题 9：地址映射

见下图



思考题 10：有本事就把我找出来！

在 kernel/src/memory/kvm.c 中，为了提高效率，我们使用了内联汇编来填写 页表项，同时给出了相应的 C 代码作为参考。如果你曾经尝试用 C 代码替换 内联汇编，编译后重新运行，你会看到发生了错误。事实上，作为参考的 C 代 码中隐藏着一个小小的 bug，这个 bug 的藏身之术十分高超，以至于几乎不影响你对 C 代码的理解。聪明的你能够让这个嚣张的 bug 原形毕露吗？

pframe_addr 是无符号类型，它的值永远大于等于 0，所以 for 循环无法退出，出现错误。

思考题 11：暗藏杀机的分页机制（这 5 个问题都有一些难度哦~）

问题一：

因为这里定义的 x 生成的地址是虚拟地址，超过了物理地址的界限，报错 0xc014a000 outside of the physical memory。

而 k vm.c 中的虚拟地址都经过了 va_to_pa 的转换，在物理地址范围之内。

问题二：在刚刚调用 init_page()的时候，分页机制并没有开启。但

通过 objdump 查看 kernel 的代码，你会发现 init_page()函数在 0xc0000000 以上的地址，为 什么在没有开启分页机制的情况下调用位于高地址的 init_page()却不会发生错误?

进行反汇编后，其地址如下:

c01003d6: e8 65 09 00 00 call c0100d40 该 call 指令的 opcode 为 e8，实现的是跳转到：该条指令的下一条指令的首地址+偏移量的位置。由于未进行寻址，故不需要进行虚实地址转化。

问题三：

- 你已经画出 init_page()函数中创建的映射了，这个映射把两处虚拟地址映射到同一处物理地址，请解释为什么要创建这样一个映射。具体地，在 init_page()的循环中有这样两行代码：

```
pdir[pdir_idx].val = make_pde(ptable);
pdir[pdir_idx + KOFFSET / PT_SIZE].val = make_pde(ptable);
```

分页的环境下，在没有初始化页表时，0 ~ 128M 的虚拟地址到物理地址的映射相当于一个简易的页表，使得高位的地址可以通过该虚拟地址（即经过 va_to_pa）访问到物理地址，从而进行初始化页表的操作。

问题四：在 init() 函数中，我们通过内联汇编把 %esp 加上 KOFFSET 转化成相应 的虚拟地址. 尝试把这行内联汇编注释掉，重新编译 kernel 并运行，你会 看到发生了错误. 请解释这个错误具体是怎么发生的。

init_mm()函数执行退出时。该函数将 nemu 映射到了高位地址并且将之前的 PDE 全部置为无效，此时返回 main.c 时，栈中保存的返

回地址需要经过虚实转换，可由于页面被置为了无效，所以报错。

问题五：在 init() 函数中，我们通过内联汇编间接跳转到 init_cond() 函数. 尝试 把这行内联汇编注释掉，改成通过一般的函数调用来跳转到 init_cond() 函数，重新编译 kernel 并运行，你会看到发生了错误. 请解释这个错误具体是怎么发生的。

查看汇编代码，直接调用此函数时，nemu 运行在物理地址上，由于在 init_mm 中将之前的 PDE 都置为无效，所以在 loader()函数寻址时页面无效，导致报错。

实验遇到的问题、思考、解决办法（可以不填写）

问题一：

对 GDT 一直看不很明白，以为数组的元素大小也需要跟处理器位数一样，最大是 32 位，对做实验造成了一定影响。

思考与解决办法：

通过查阅资料，了解到实际上 GDT 的元素就是 64 位的段描述符，不过是分成了两个 32 位的结构体。

问题二：

对分段很不理解。

思考与解决办法：

指导书内容进行了相应的整理，并上网查阅资料。了解到了许多知识，如：CPU 还需要记录 GDT 的基地址，这通过设置一个 GDTR 的寄存器来实现，该寄存器有 48 位，前 32 位类似于指针存放 GDT 的首地址，后 16 位记录 GDT 的长度。

实验心得（可以不填写）

第一，我要加强查找文献、整理文件、阅读文献的能力，尤其是加强阅读英文文献的能力。

第二，学会自己解决问题，可以通过网络搜索解决大部分的问题。

第三，遇到不会的问题，可以请教老师、与同学交流，增进对知识的理解。

第四，面对大量代码时，首先应该梳理框架，再抓细节。

第五，指针最大位数受 CPU 位数影响，所以才需要 GDT 进行段描述符的存储和读取。