Lab2 要点回顾

2008-03-19

Lab2要点

- 理解x86内存映射机制的细节
- 了解实习内核的内存布局
- 理解并实现boot_alloc()中简单物理内存管理方案，理解其中需要对齐的原因
- 理解并实现page_*()中基于页面的内存管理方案，理解并部分实现4M页面功能*
- 理解内核启动过程中关于内存映射的操作流程及内核自身映射地址变化过程

Contents

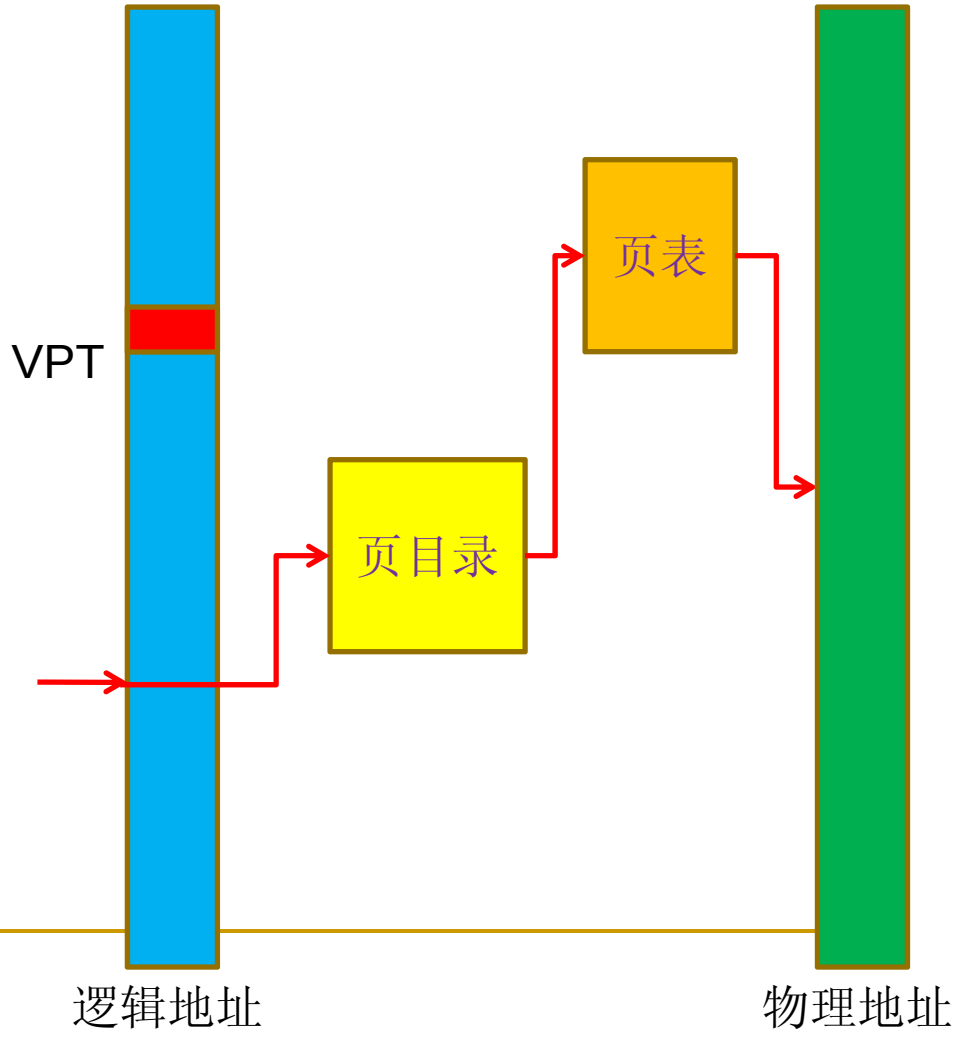
- 答疑：**VPT**中的递归映射
- 内存映射机制和实习中的使用
- 内存对齐的作用
- 内核位置随内存映射的变化
- **Challenge** 要点
- 其他需要掌握的内容
- **Q&A**

Virtual Page Table 映射

- 在kern/pmap.c
  - Pgdir[PDX(VPT)] = PADDR(pgdir) | ……
  - 为pgdir建立了VPT位置指向自身的映射
- 无VPT映射时访问修改页表项方式
  - Pgdir[PDX(va)]得到页表的物理地址，转换为逻辑地址，再根据逻辑地址取PTX(va)的偏移量，得到pte
- 映射VPT后访问方式
  - 在逻辑地址VPT开始的4M空间中，即是所有4G空间的pte的排布，根据VPT[PDX(va)*4K+PTX(VA)]即可直接访问一个pte
- 将需要代码实现的逻辑用CPU的mmu实现，提高效率，方便代码编写

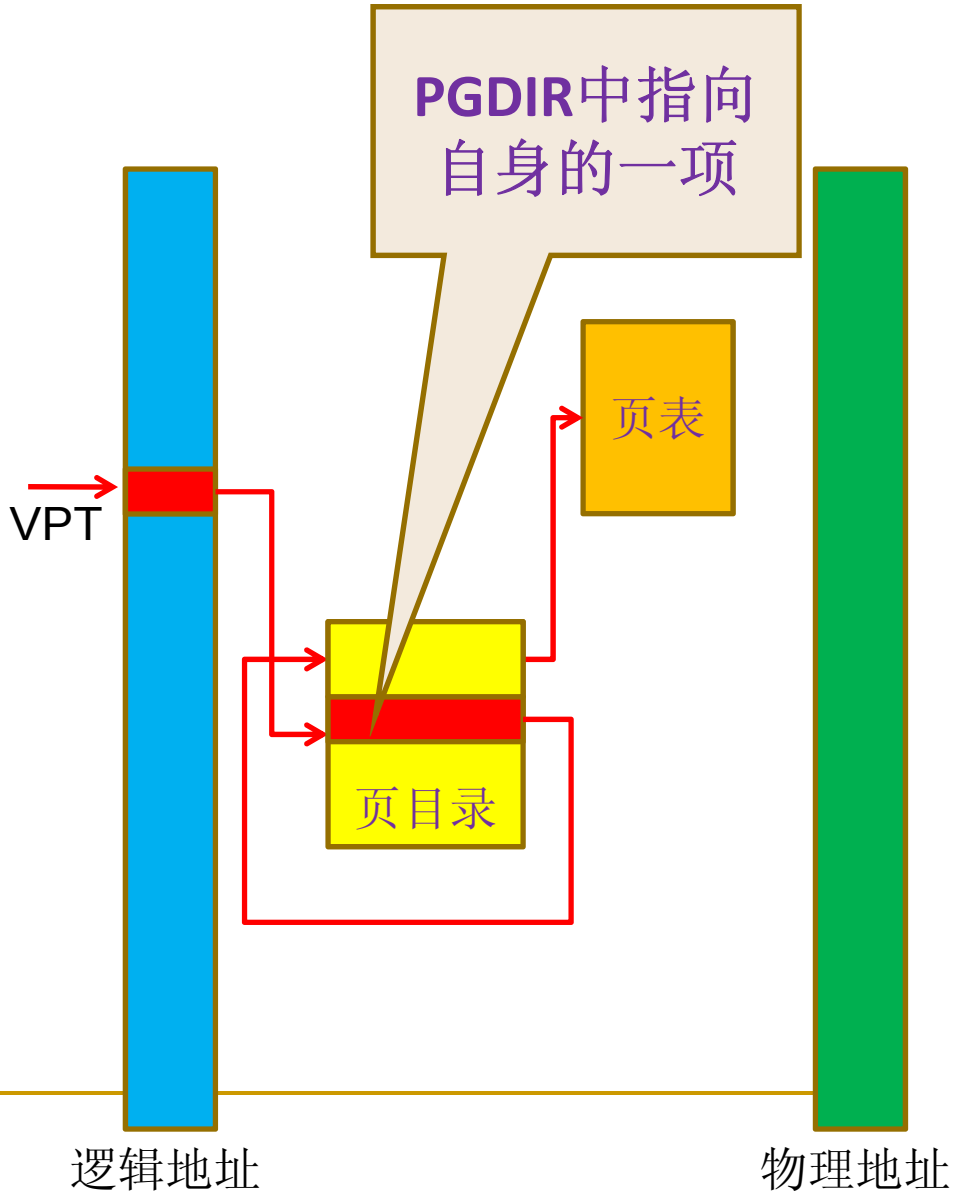
Virtual Page Table 映射

- 非**VPT**区域内存映射步骤
- 正常的**2**级页表，三次查找的寻址过程



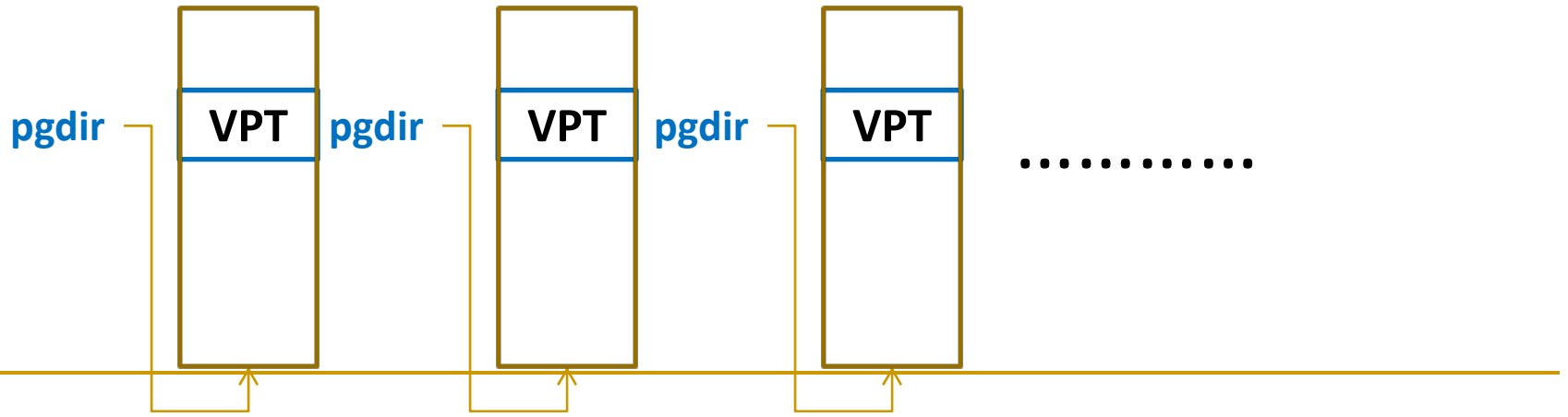
Virtual Page Table 映射

- VPT区域内存映射步骤
- 2级页表的三次查找的寻址过程中，第一次在页目录中转了一圈，故访问到的内存是页表项的内容



Virtual Page Table 映射

- **Recursively?**
- **Kern/pmap.c第161行**
  - **“Recursively insert PD in itself as a page table, to form a virtual page table at virtual address VPT”**
- 此处递归是指一个页目录中的一项指向自己

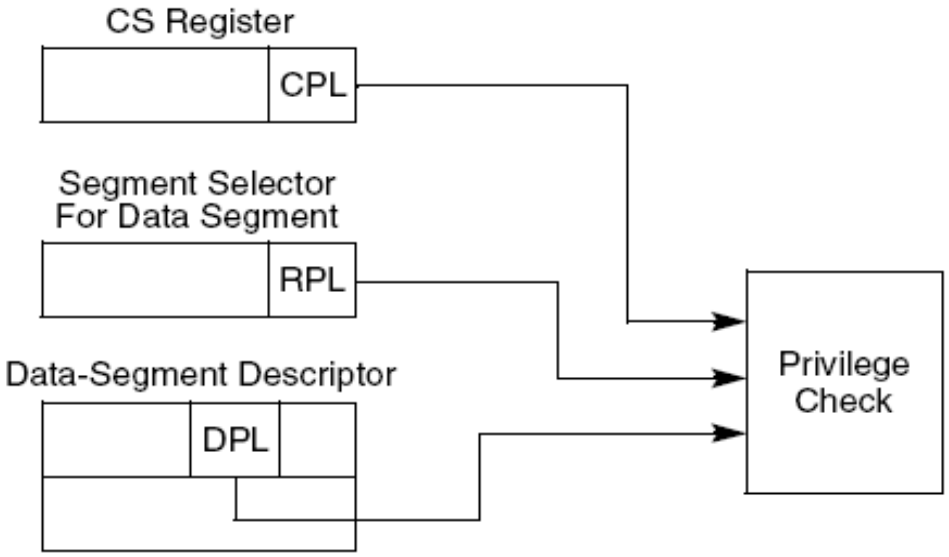


内存映射机制

- 对权限的检查
  - 段：写，执行，CPL是否够访问该段
  - 页：写，CPL是否能访问该页面
- 映射精度
  - 段：起始地址精确到字节，段大小20bit
    - 精确到字节，表示范围1M
    - 精确到4K，表示范围4G
  - 页：边界对齐到4K
- 实习用到功能
  - 段：局部用来调整地址偏移量，最后相当于被关闭
  - 页：页存在、写、CPL

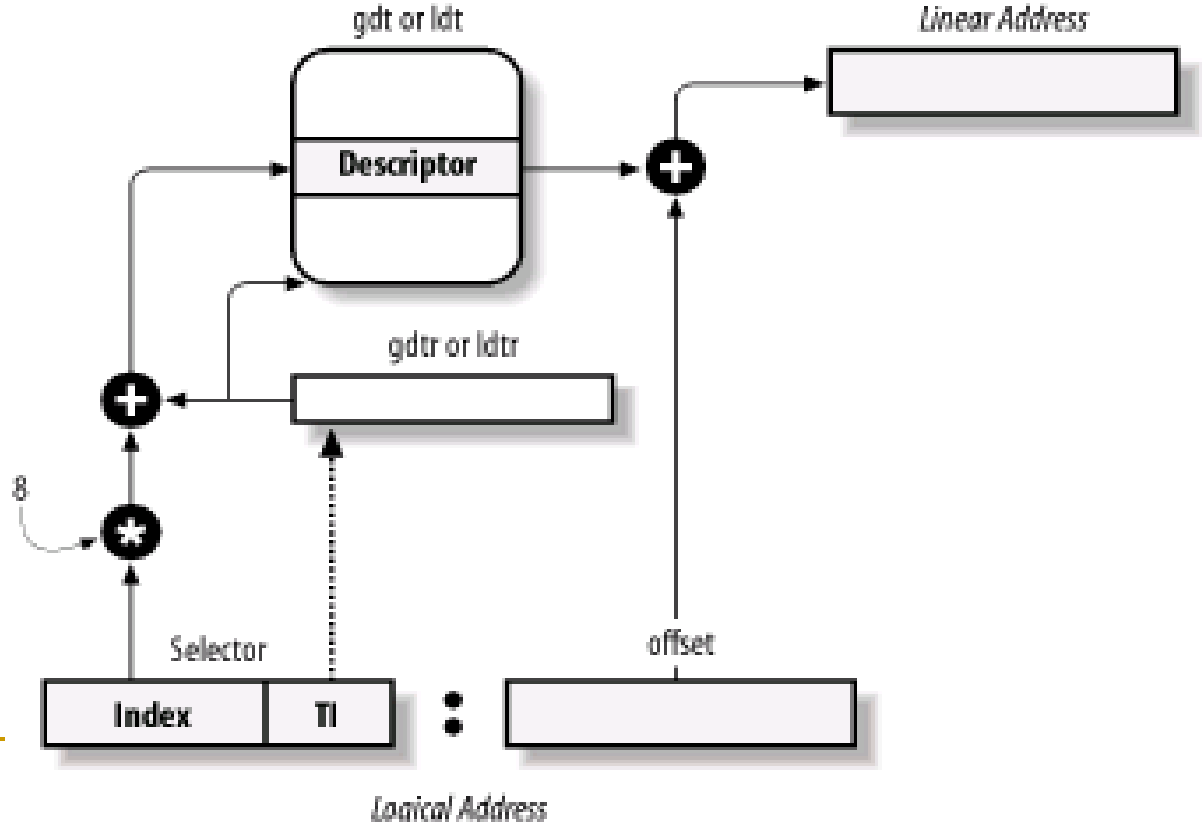
段页映射中的权限检查

- 权限级别检查
  - **CPL** 当前所执行代码的特权级（内核态/用户态）
  - **RPL** 段选择子的特权级
  - **DPL** 段描述符所代表的内存段的特权级



段映射

- 逻辑地址生成线性地址
- 区分全局段描述符**gdtr**和本地段描述符**ldtr**

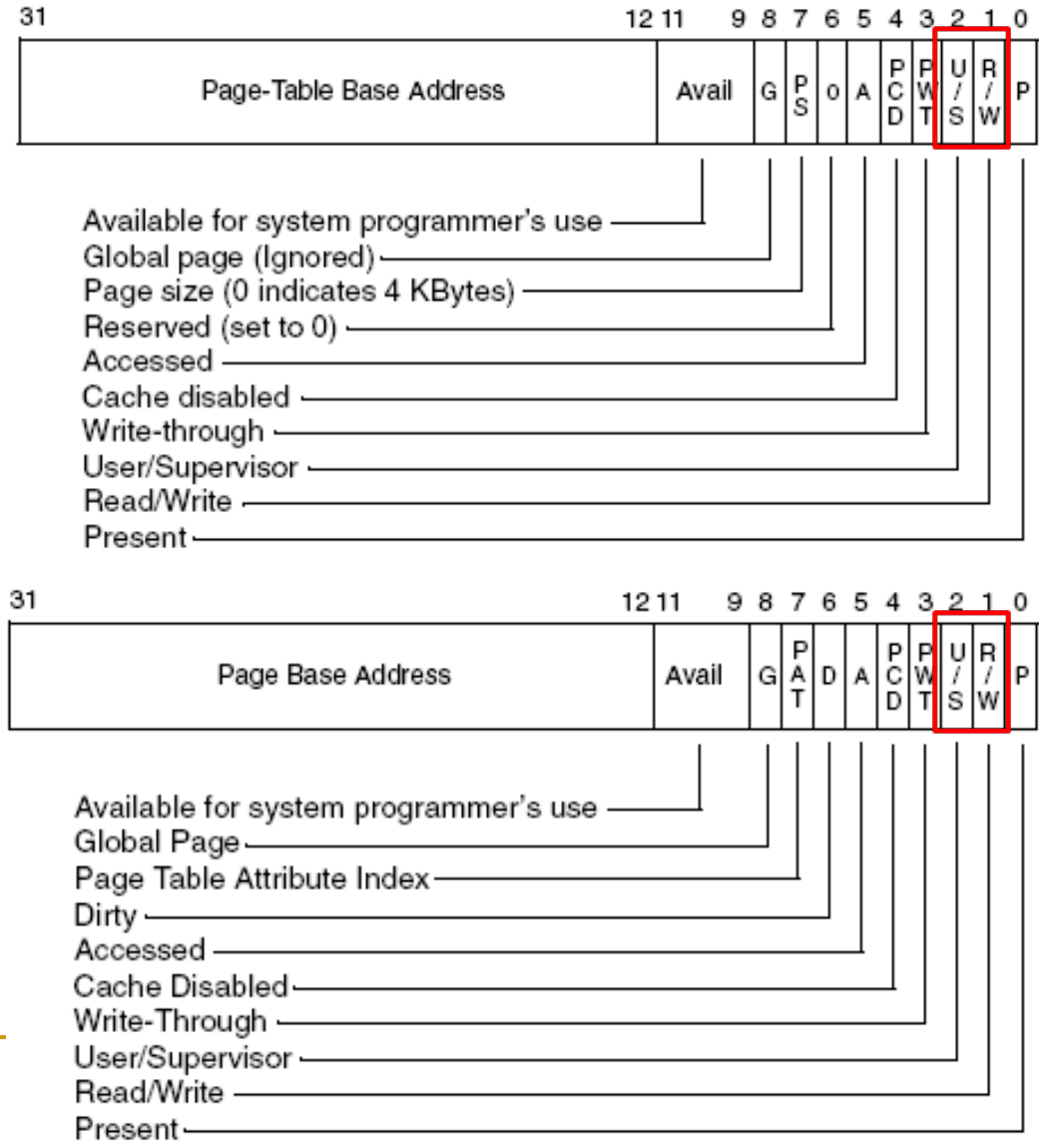


段映射权限检查

- 段选择子CS
  - 15-3bit：选择的段在段描述表中的位置
  - 2bit：使用GDTR还是LDTR
  - 1-0bit：当前程序的CPL
- 当前代码的CPL执行权限由CS的CPL位确定，用户程序试图修改时会发生保护错误(GPF)

页映射权限检查

- 红色框中即为页映射权限设置位
  - **U/S为1表示用户可访问(CPL1,2,3)，为0表示只有内核可访问(CPL0)**
  - **R/W表示读写权限，1表示可写，0表示可读**



内存映射权限检查步骤-段映射

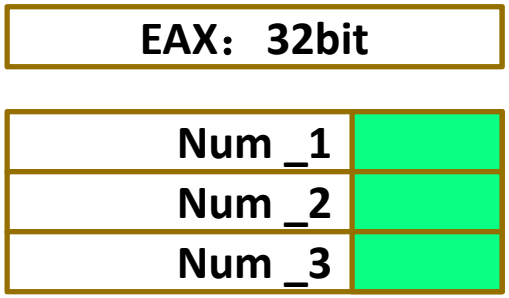
- 得到所访问段的选择符?s(cs/ds/ss/gs)*和地址addr
- 确定是否有访问段的权限
  - 比较当前CS中的CPL是否高于等于*
    - 所用来访问的段选择符中的RPL
    - 所访问的段描述符表中定义的DPL
  - 两者同时满足，段访问通过，否则产生一般保护错误(General Protection Fault:GPF)
- 根据?s:addr生成线性地址

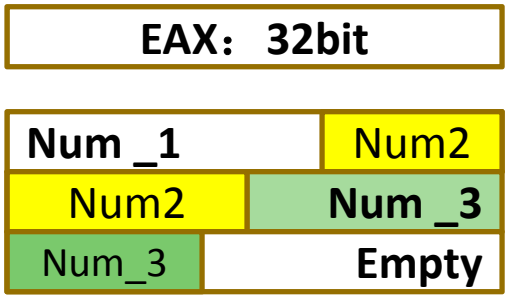
内存映射权限检查步骤-页映射

- 得到段映射生成的线性地址
- 确定是否有访问页面的权限
  - 确定当前CPL是否足以访问页对应的页目录项
    - 页目录项PTE_P为0，则页面不存在，无法访问
    - 页目录项PTE_U为0，则只有CPL0可以访问
    - 页目录项PTE_W为0，则禁止写操作
  - 若页目录项PTE_PS为0，则继续检查页表项，否则通过
    - 页表项检查PTE_P, PTE_U, PTE_W方式和页目录项相同
  - 任何一步确定不能访问后，产生页错误(Page Fault: PF)
- 根据物理地址访问内存

boot_alloc() & align

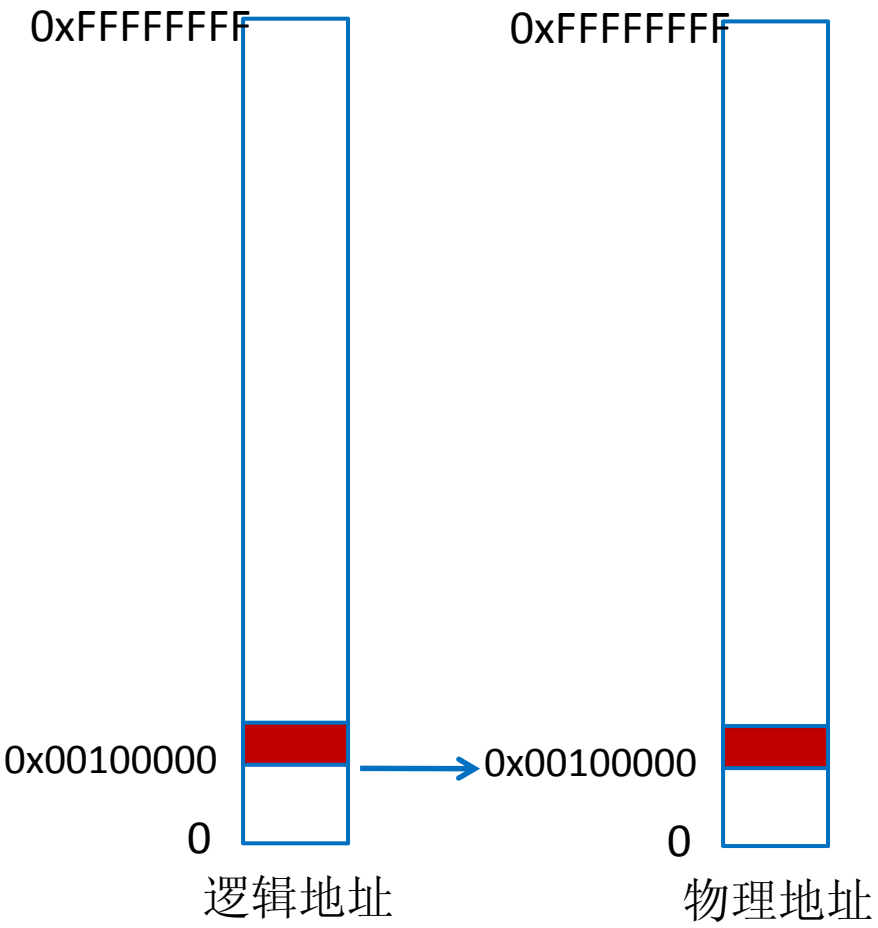
- 为什么要对齐
  - 内核数据结构要求对齐
    - **Pgdir, pgtable等硬件上即要求4K边界对齐**
  - 对其可以提高执行效率
    - 数据以**CPU**数据宽度排列以降低空间利用率为代价提高速度





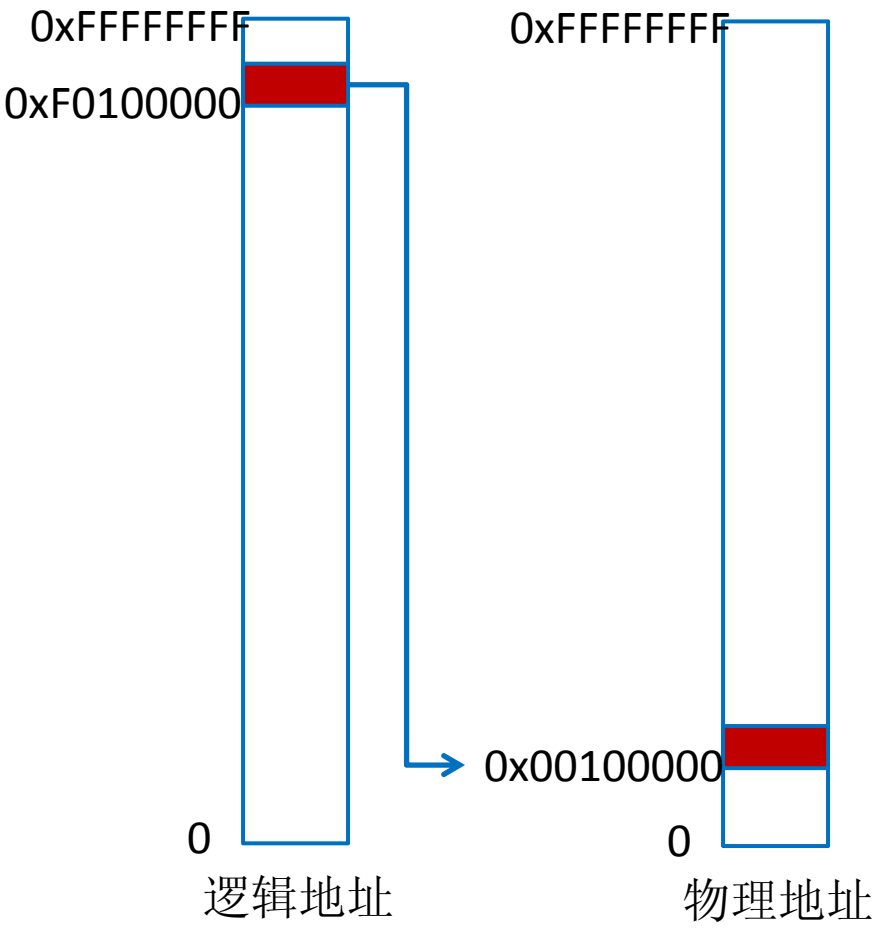
内核的位置-1

- **BootLoader**加载时
  - 物理地址**1M**，逻辑地址**1M**
- 内核的链接地址是**0xF0100000**，但加载地址是**0x00100000**，故在跳转到内核时，需将**ELF**格式中提供的入口地址高位清**0***



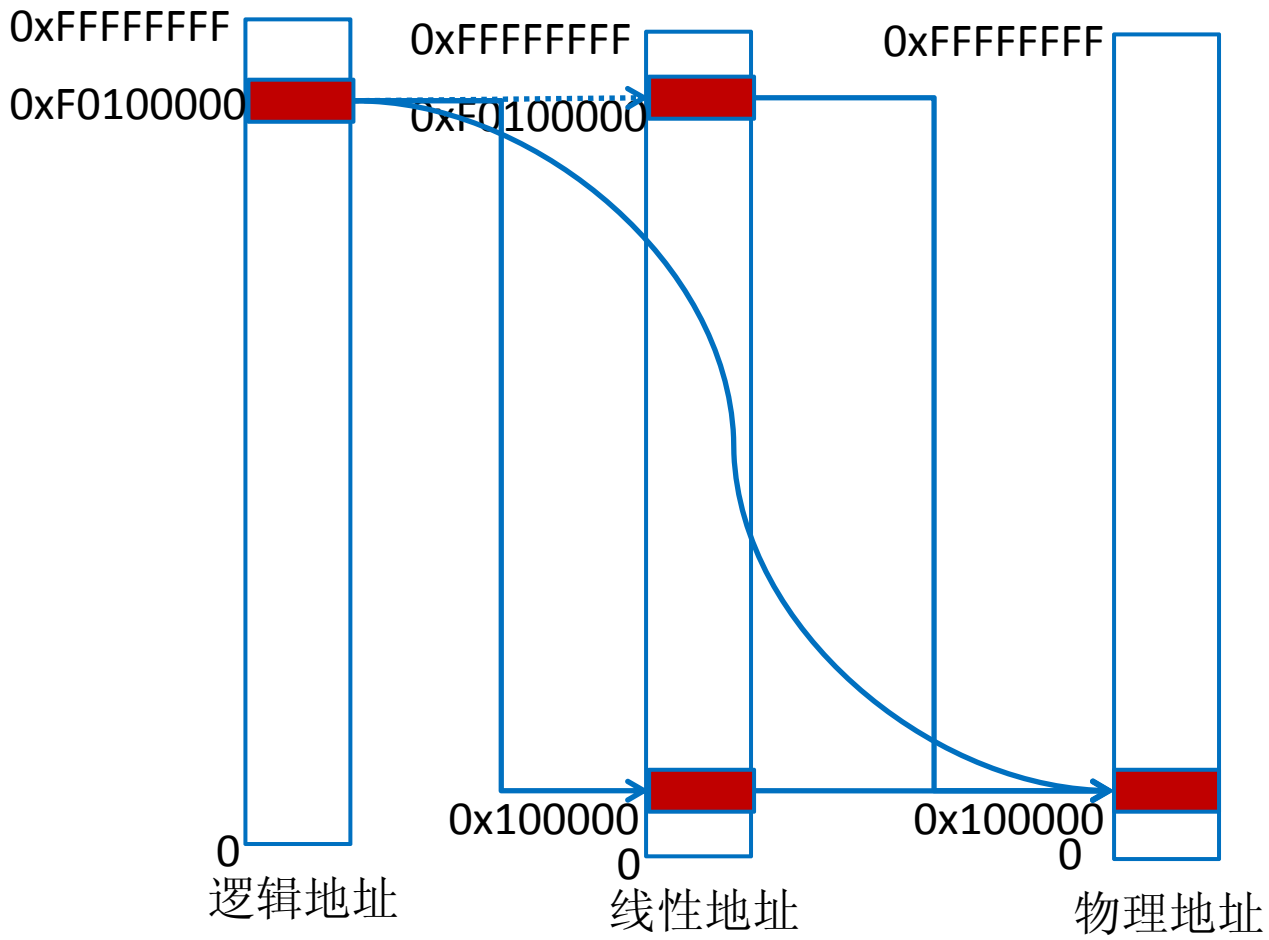
内核的位置-1

- 内核启动后加载自己的**GDT**
  - 只开段映射
  - 段起始地址**-KERNBASE**
  - 物理地址**0x00100000**，逻辑地址**0xF0100000**
- 使用**ljmp**加载新的段映射



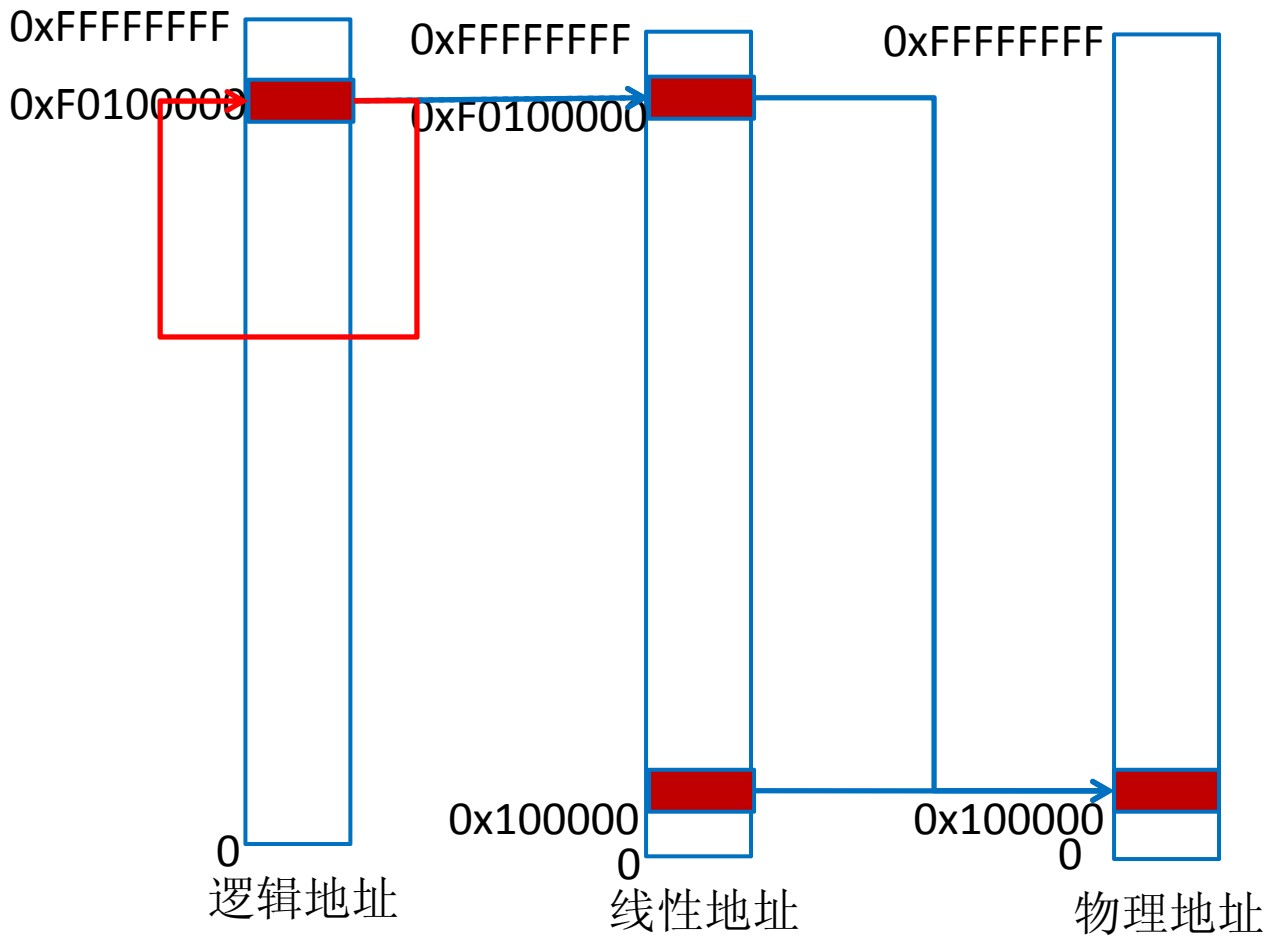
内核的位置-2

- 映射物理内存
- Pgdir[0]=pgdir[PDX(KERNBASE)]，重复映射一下
- 打开页映射
- 加载新的gdt
- 保证打开页映射时内核正常执行(此时新段映射尚未生效)



内核的位置-3

- lgdt并不代表新的段映射已经生效
- ljmp cs, ip才会使新的段映射实际生效
- pgdir[0]=0去除冗余的页映射



关于ljmp

- ljmp的作用
  - 加载cs、ip的远跳转
  - 内存映射配置切换时，使用ljmp重载cs使更改生效

关于内存权限放宽

- 当内核访问内存时发生错误，同时无法知道错误原因时：**(Lab2, Lab3)**
  - 尝试放宽内存权限：将用户不可读的设为可读，原先只读的设为可写，以便使用**printf**法定位错误原因
- 原因
  - 在**Lab3**后半部分处理**PageFault**和**GPF**之前，发生访存错误产生的**PF**和**GPF**无法被处理，导致无法知道错误原因
  - 在**pagefault**处理功能完成后，可以根据出错地址、出错代码(页面不存在/用户访问系统页面/写操作)以及出错代码**ip**等信息定位错误

Challenge 要点

- **Challenge1**
  - 注意**4M**页面的配置
  - 在**boot_map_segment()**中映射**4M**页面时，要考虑到**4M**页面是以**4M**为页框边界的，若映射区域起始、中止位置不以**4M**为边界，则需要考虑好**4K**的“零头”的处理问题

Challenge 要点

- **Challenge2**
  - 理解内核控制台新命令的添加方式
    - 命令列表：

      kern/monitor.c: struct Command commands[]
    - 命令函数原型：

      Int mon_*(int argc, char **argv, struct Trapframe *tf)
  - 注意：显示逻辑地址内存内容时，需要先检查地址是否已映射物理内存，避免出错

Challenge 要点

- **Challenge3**
  - 理解**x86**硬件内存管理机制
  - 理解实习内核的内存管理机制
  - 参考**Linux**内核源码阅读中，内核管理内存的方式
  - 设计使用户能使用完全的**4G**逻辑地址空间的内存管理方案，并写入文档

Challenge 要点

- **Challenge4**
  - 注意**4M**页面的页框边界问题
  - 尽可能有效的利用空闲内存，在一种页面大小空闲内存耗光后，能将**4M**页面分拆以相应**4K**页面的申请，也能将可能的**4K**页面合并成大页框以相应**4M**页面申请。可以考虑参考**Linux**的伙伴系统**(**见源码阅读**ppt)**
- **Challenge5**
  - 实现内核控制台命令更能，参考**Challenge2**

其他需要掌握的内容

- 内存大致布局
  - **KERNBASE**的位置，高于和低于**KERNBASE**的逻辑地址使用情况
- 重要的常量、宏、变量的作用
  - **KERNBASE;ROUNDUP(addr, align); ROUNDUP(addr, align); PTE_P; PTE_PS; PDX(addr); PTX(addr); PTE_ADDR(); KADDR(); PADDR()**
  - **etc…**
- 使用指针、地址时，要时刻清楚那是物理地址还是逻辑地址

重要的宏/变量/函数

- **KERNBASE**
  - 内核逻辑地址的起始点。从**KERNBASE**到**4G**的逻辑地址映射了**0-256M**的物理内存，以方便内核直接访问
- **KADDR(pa)/PADDR(va)**
  - 由于**KERNBASE**以上逻辑地址和物理地址有线性映射，设置这两个宏来方便其转化
  - 只对**KERNBASE**以上的逻辑地址有效，程序会自动**assert**错误的使用

重要的宏/变量/函数

- **ULIM**
  - 用户态程序可以访问地址的界限，更高的内存用户不可读
  - 一般用来方便判断用户访存是否超界
- **UTOP**
  - 用户有写权限的地址界限。**UTOP**和**ULIM**之间是用户只读的内核数据结构，如**UVPT**，**UPAGES**

重要的宏/变量/函数

- **VPT/UVPT**
  - 当前用户进程/内核的页表项目映射的位置，其中每个**4K**页面对应一个页目录项中的一个页表
  - **UVPT**是为用户程序只读访问**VPT**而做的映射
- **UPAGES**
  - **Page**结构数组在内存中的映射，其中每个**page**结构记录了一个物理页面是否被使用以及使用次数、在空闲页链表中使用的链接等信息

重要的宏/变量/函数

- **PDX(va)/PTX(va)**
  - 页映射时，根据逻辑地址计算va所在页目录项的序号和页表项序号的红
- **PTE_ADDR(va)**
  - 抽取页表项中地址部分内容的宏。由于页表项中除了高20bit保存地址外，低12bit保存参数，设置此宏可方便代码编写
  - 使用4M页面的话可以类似的实现PDE_ADDR(va)来简化代码

重要的宏/变量/函数

- **page2ppn/page2pa/pa2page**
  - 与物理页面一一对应的**Page**结构中没有保存其对应的物理地址，而是通过其在**page**结构数组中的序号来进行物理页面寻址
  - 通过**page2pa**和**pa2page**可以得到**page**结构和物理页面地址的相互转化
- **Page2kva**
  - 在**Page2pa**后，我们希望能直接访问其逻辑地址，故一个**KADDR**处理后，即得到了一个**page**结构对应的**kva**

- 以上只是**Lab2**中重要的宏、变量等
- 了解整个内核代码各个部分对编写代码有很大帮助，建议大家多读代码，分析其各个部分的作用

The end

Q&A