Lab2 简介

2008-03-05

Lab2时间安排

- Lab2时间：3月5日至3月25日
- 第一周
  - Part1物理内存管理，Part2虚拟内存管理
- 第二周
  - Part3内核地址空间设置及调试
- 第三周
  - Challenge、代码调试和文档

Lab2实习题目

- **Exercise1~6**
  - 必做
- **Questions**
  - 必做，写入文档
- **Challenge1: 4M页管理内核内存**

  Challenge2: 控制台命令扩展
  - 必做
- **Challenge3: 用户地址空间扩展**
  - 必做，写入文档

Lab2实习题目(续)

- **Challenge4: 4M页内存管理设计与实现**
  - 选做
- **Challenge5: 控制台命令扩展(页面申请/释放控制)**
  - 必做

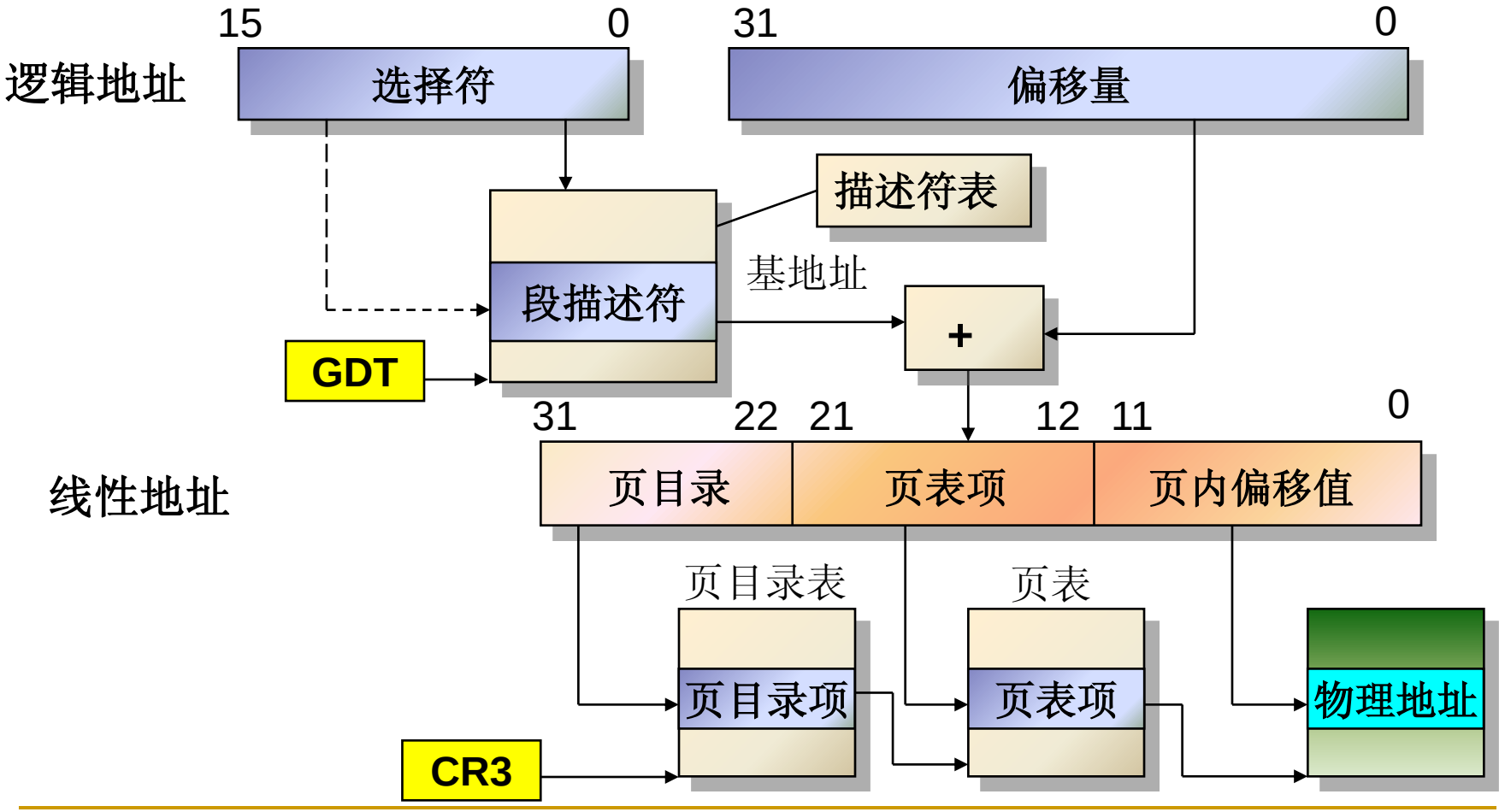
实习要求

- 理解关于内存映射和管理的原理和要求
  - 进行代码填空，完成函数功能并通过测试
    - 物理页面管理
    - 虚拟页式内存管理
  - 在文档中对要求文档中的问题进行回答

Lab2内核内存管理的任务和方式

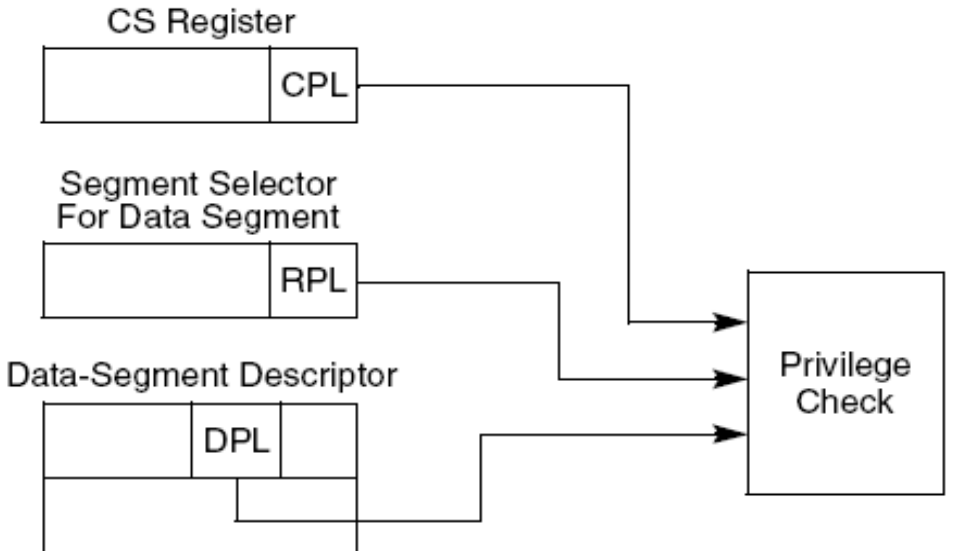
- 内核与用户程序之间的保护
  - 段映射特权级别
    - 控制访问所需的特权级别(ring0123)
  - 页映射特权级别
    - 控制读、写、用户态可否访问
- 用户程序之间的保护
  - 不同进程使用同一个段，不同的页目录(本实习)
- 提供内存控制API

段页式内存映射机制


15
0
31
0
逻辑地址
选择符
偏移量
描述符表
段描述符
基地址
+
GDT
31
22
21
12
11
0
线性地址
页目录
页表项
页内偏移值
页目录表
页表
页目录项
页表项
物理地址
CR3

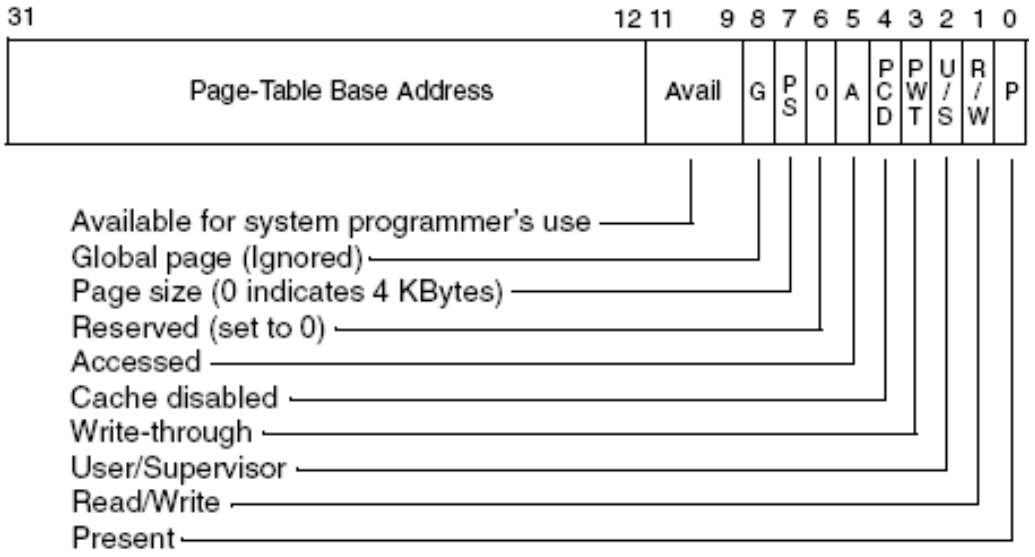
段页映射中的权限检查

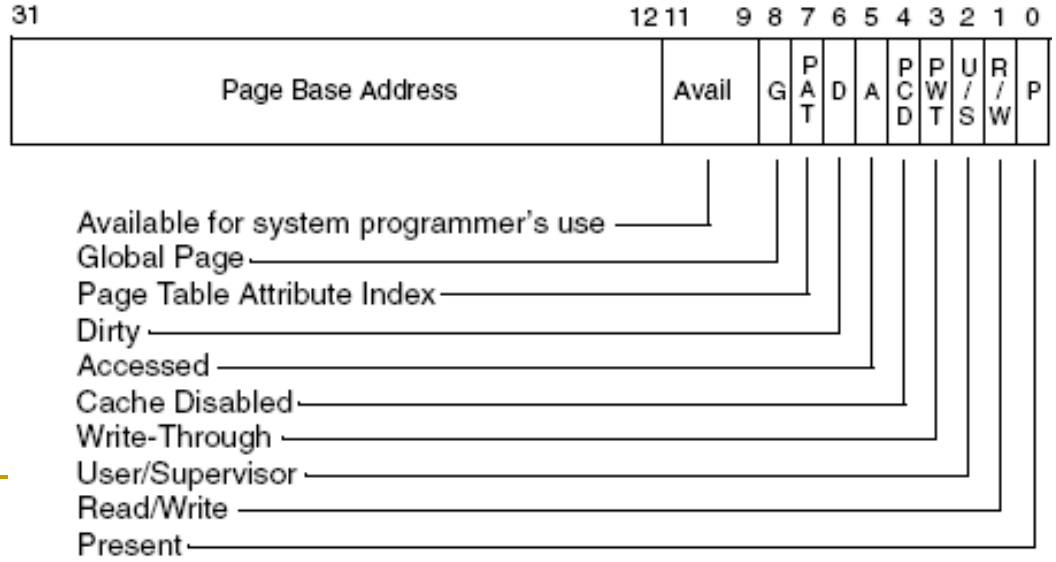
- 权限级别检查
  - **CPL** 当前所执行的代码的特权级（内核态/用户态）
  - **RPL** 段选择子的特权级
  - **DPL** 段描述符所代表的内存段的特权级



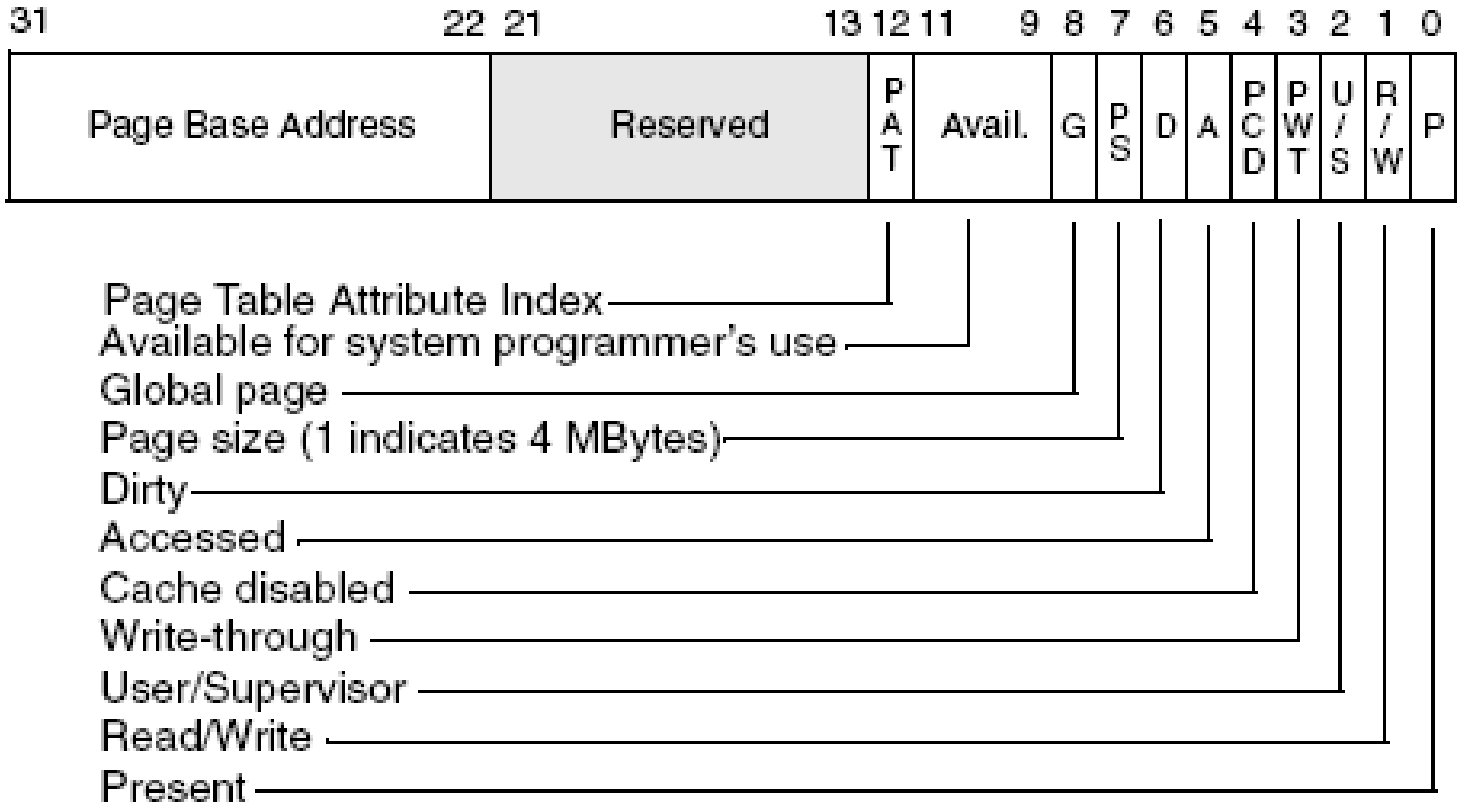
页映射中的数据与意义

- 页目录项
  - **4M页Challenge**
  - 调试时注意检查此处的权限
- 页表项





4M页时页目录项结构

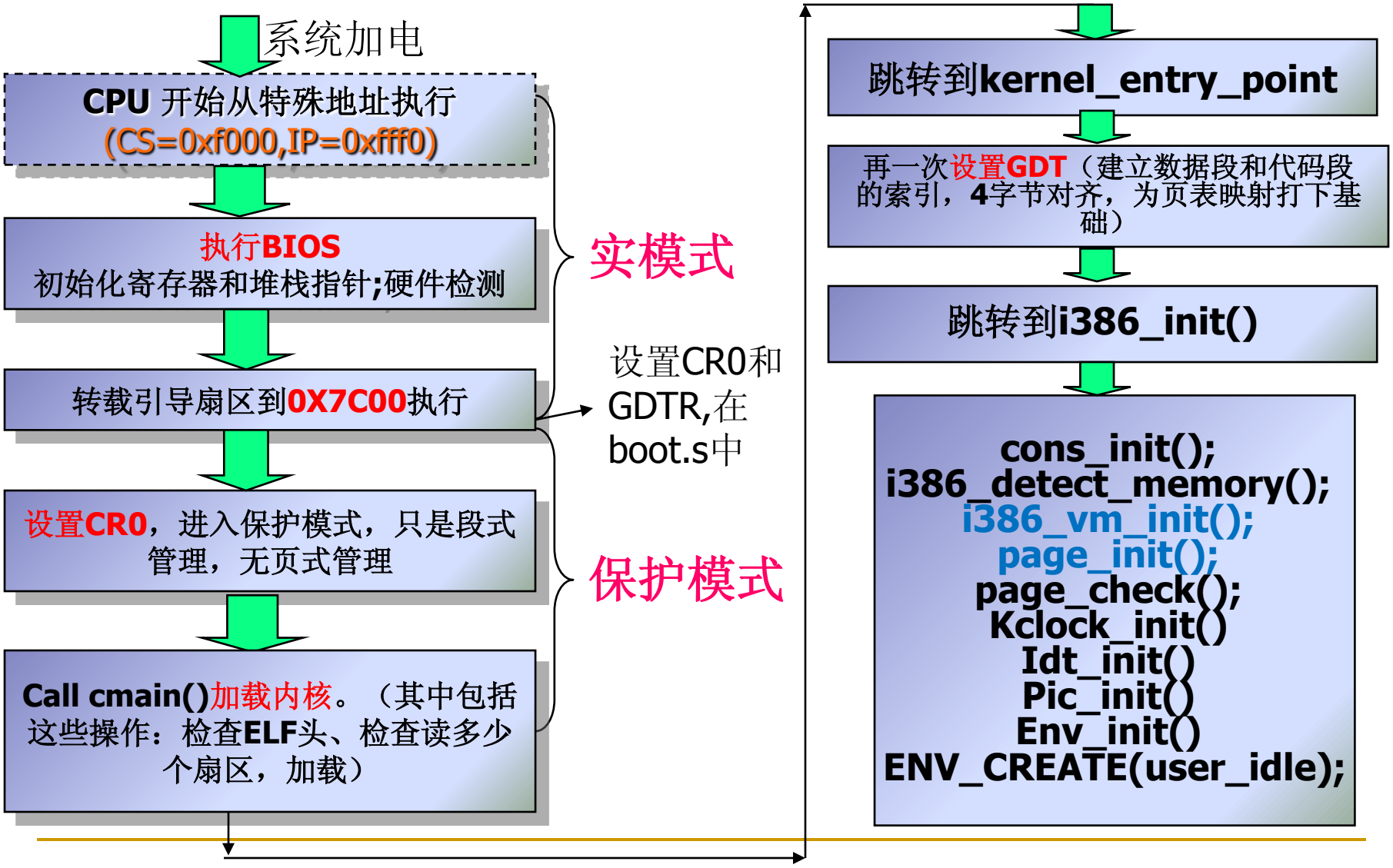


- 结构图摘自Intel手册，详细参考可找参考资料

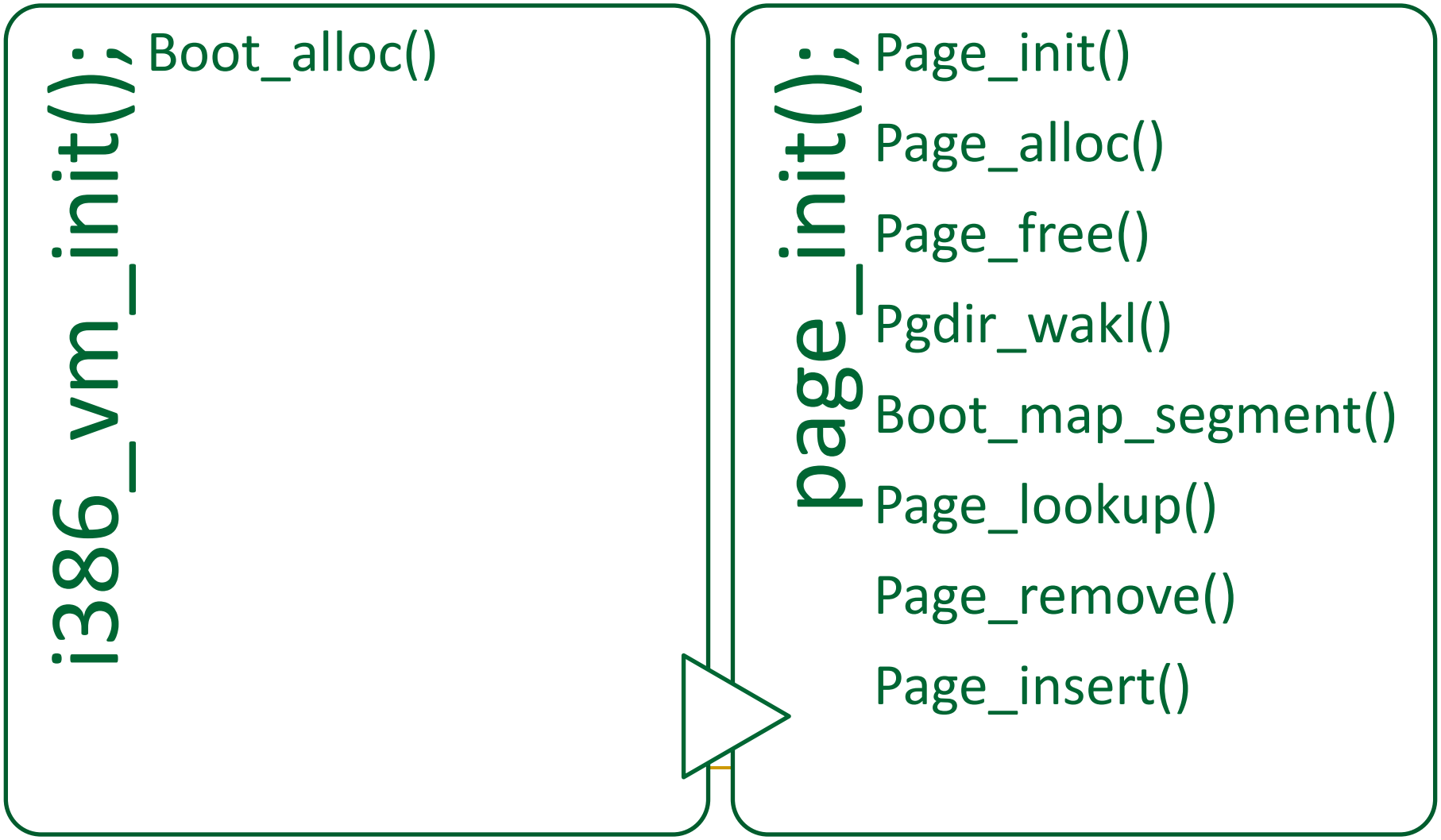
MMU - TLB

- TLB（Translation Look-ahead Buffer）
  - 将页表项缓存在CPU中，加快地址转换速度
  - 页表项和页目录项的G位使得CR3改变时页表项或页目录项仍然缓存在CPU中
- 如何设置哪些页表常驻TLB会使系统效率提高？
- 何时需要使用tlb_invalidate(Pde *, u_long)清空TLB？

内核内存管理流程


系统加电
CPU 开始从特殊地址执行
(CS=0xf000,IP=0xfff0)
执行BIOS
初始化寄存器和堆栈指针;硬件检测
转载引导扇区到0X7C00执行
设置CR0，进入保护模式，只是段式管理，无页式管理
Call cmain()加载内核。（其中包括这些操作：检查ELF头、检查读多少个扇区，加载）
实模式
设置CR0和GDTR,在boot.s中
保护模式
跳转到kernel_entry_point
再一次设置GDT（建立数据段和代码段的索引，4字节对齐，为页表映射打下基础）
跳转到i386_init()
cons_init();
i386_detect_memory();
i386_vm_init();
page_init();
page_check();
Kclock_init()
Idt_init()
Pic_init()
Env_init()
ENV_CREATE(user_idle);

所需实现的函数(部分)


i386_vm_init();
Boot_alloc()
page_init();
Page_init()
Page_alloc()
Page_free()
Pgdir_wakl()
Boot_map_segment()
Page_lookup()
Page_remove()
Page_insert()

内存布局

```
/* Virtual memory map:                                Permissions
 *                                                    kernel/user
 *        4 Gig --> +----------------------------+
 *                  |  Remapped Physical Memory  | RW/--
 *     KERNBASE --> +----------------------------+ 0xf0000000
 *                  | Cur. Page Table (Kern. RW) | RW/--  PTSIZE
 *VPT,KSTACKTOP --> +----------------------------+ 0xefc00000    --+
 *                  |        Kernel Stack        | RW/--  KSTKSIZE |
 *                  | - - - - - - - - - - - - - -|                 PTSIZE
 *                  |     Invalid Memory (*)     | --/--           |
 *         ULIM --> +----------------------------+ 0xef800000    --+
 *                  | Cur. Page Table (User R-)  | R-/R-  PTSIZE
 *         UVPT --> +----------------------------+ 0xef400000
 *                  |         RO PAGES           | R-/R-  PTSIZE
 *       UPAGES --> +----------------------------+ 0xef000000
 *                  |                            |
 *                  |      Empty Memory (*)      |
 *                  |                            |
 *            0 --> +----------------------------+
 */
```

实习代码优化

- 实习的代码有些地方有优化的余地
  - 页目录、页表选重要的设为**GlobalPage**
  - 使用**4M**页
  - 后面实习中也有能优化的地方
  - 其他我们未想到的优化
- 在文档中说明优化的位置、方法、原理及可能做出的测试方法

实习要求1

- 将lab1的解答放入lab2
  - 取得lab1修改内容
    - diff -u -r lab1 lab1-change > lab1-changes.patch
  - 将lab1的修改patch到lab2
    - cd lab2
    - patch -p1 -u < ../lab1-changes.patch
  - 将*.rej中未加入的更改手动加入相应文件

实习要求2

- **Exercise1：**
  - 扩展Lab1中的Stack Backtrace功能，通过调用stab_binsearch 与read_eip()实现debuginfo_eip()函数，使原先显示eip的位置变为所在的函数名称和偏移量

```
Stack backtrace:
kern/monitor.c:74: mon_backtrace+10
  ebp f0119ef8  eip f01008ce  args 00000001 f0119f20 00000000 00000000 2000000a
kern/monitor.c:143: monitor+10a
  ebp f0119f78  eip f01000e5  args 00000000 f0119fac 00000275 f01033cc fffffffc
kern/init.c:78: _panic+51
  ebp f0119f98  eip f010133e  args f01033ab 00000275 f01033cc f0103473 f01030bc
kern/pmap.c:711: page_check+9e
  ebp f0119fd8  eip f0100082  args f0102d20 00001aac 000006a0 00000000 00000000
kern/init.c:36: i386_init+42
  ebp f0119ff8  eip f010003d  args 00000000 00000000 0000ffff 10cf9a00 0000ffff
```

实习要求2

- **Exercise2：**
  - 完成函数boot_alloc()、page_init()、page_alloc()、page_free()，实现对物理内存页面的管理
- **Exercise3：**
  - 阅读Intel手册，了解段式映射在保护模式下的使用

实习要求3

- **Exercise4：**
  - 阅读Bochs手册中关于内存内容显示的功能介绍，区分逻辑/线性/物理地址，熟练使用命令来验证自己的程序
- **Exercise5：**
  - 完成函数pgdir_walk(), boot_map_segment(), page_lookup(), page_remove(), page_insert()等，实现物理页面排布，并通过page_check()的测试

实习要求4

- **Exercise6：**
  - 完成i386_vm_init()，通过代码测试

- 注意按时、按名称提交到所属的助教邮箱

重要代码、数据结构(1)

- 在内核虚拟内存布局中，需要设置内核数据结构以及堆栈的映射，注意以下宏的定义和使用
  - **boostack**
  - **Kernbase**
  - **VPT**
  - **UENVS(user readable)**
  - **UPAGES(user readable)**
- 注意**inc/memlayout.h**中的内存布局

重要代码、数据结构(2)

- 二级页表映射的建立
  - 阅读mmu.h文件，熟悉一些PDX和PTX的宏的定义
  - 阅读kern/pmap.h文件，了解如下函数
    - Pa2page--得到线性地址
    - Page2pa--返回内存页表数
    - Page2ppn--验证内存页的正确性.返回page所在地址
    - PADDR—虚拟地址转换成物理地址
    - KADDR—物理地址转换成虚拟地址
    - Page2kva--把物理地址转换为虚拟地址

重要代码、数据结构(3)

- Page_init()初始化函数，该函数的编写重点是要找到内核区域

  注：该函数以后还需要修改
- 如果熟悉链表操作，那么编写这些函数不是一个很困难的事情

  注：在编写时，需要把页面访问的权限放大，这样会减少将来lab调试的很多麻烦。必要时设置用户态可读

重要代码、数据结构(4)

- 注意kern/pmap.c中
  - pgdir[0] = pgdir[PDX(KERNBASE)];
  - pgdir[0] = 0;

  作用分析
  - 内核的Link Address是如何与Load Address匹配的
  - 在此之前两个地址是否相等？
  - 如果没有，代码中做了哪些工作来解决？
  - 在内核两个地址不相等时，做什么操作会发生问题？

Q&A