

# OS Lab2 设计文档

516030910293 姚子航

## 一、概要

本文档为操作系统 lab2 设计文档，设计文档按照 lab2 文档结构组织。

## 二、物理页管理

在 linux 系统中，内核使用伙伴系统来管理物理内存，即将所有的空闲页面分为 11 个空闲链表，分别管理着 1, 2, 4, 8....1024 个连续的空闲页框。内核还将内存空间分为三种区：DMA, NORMAL, HIGHMEM 每个管理区使用不同的伙伴系统。

伙伴系统有一些优点，比如方便系统分配连续内存，较好的解决了外部碎片的问题等。但是伙伴系统也会导致内存出现碎片，内存空间的浪费等问题，而且算法较为复杂增加 overhead。为了解决以上的几个问题，linux 还引入了 slab 分配器来专门分配一些小块内存，将一些小块内存维护在一个内存池中，这样就不需要每次都和伙伴系统进行交互，提高分配效率。

在 Jos 系统中，我们没有实现伙伴系统，而是只用了一个空闲链表来管理物理内存页，内核在启动的时候使用虚拟 CMOS 检测物理内存空间大小，之后内核根据检测出的大小使用 page\_init 来初始化管理页的空闲链表，注意，在构建完成空闲链表之前，我们使用 boot\_alloc 函数在内核内存空间分配必要的内核数据结构，该函数使用了一个链接过程中引入的 end 变量来指向内核的顶端地址，内核从此处开始构建内核数据结构，值得注意的一点就是，在之前的 lab 中我们已经开启了分页机制，这意味着我们现在函数中所用的变量的地址都是高位的虚拟地址，在这个问题上我被困扰了很久。构建好了空闲链表后也就完成了第一部分的内容。

## 三、虚拟内存

在 x86 架构中，使用分段机制与分页机制来进行地址的翻译，一个虚拟地址包含了一个段选择符和一个段中偏移量，在地址翻译的过程中，我们先将 virtual address 通过段选择符与偏移翻译为 linear address，linear address 再经过分页机制翻译为物理地址。在 Jos 系统中，我们只使用分页机制来进行保护与虚拟地址翻译，段机制我们通过几个统一的，base 为 0 的段描述符来淡化它的作用。所以 linear address 就等于 virtual address。

接下来我们开始构建真正要用的内核页表，之前的页表只是一个临时使用的页表。在这个页表中我们依然将内核地址映射在高位，内核只需要将内核数据虚拟地址减去 kernbase 就可以得到其物理地址。在这个部分我们需要实现一些操作页表的帮助函数，在这些帮助函数中，我们调用第一部分完成的 page\_alloc, page\_free 等函数。注意在实现 page\_insert 函数时有一个 corner-case 需要注意，就是在我们重新将一个已经映射在某地址的一个页重新插入该地址的时候，我们需要先增加这个页的 reference count，再 remove 原来映射的这个页，这样就能避免虚拟页被错误的释放。

## 四、内核地址空间

Jos 中虚拟地址空间的布局已经在 memlayout.h 这个文件中给好，地址空间由 ULIM 这个地址分割成两部分，上面的部分是用户不可见，只有内核可以管理的区域，包括 kernbase 以上的内核空间，以及 memory-mapped IO 的空间，在 ULIM 下方，内核将一些数据结构比如物理页的空闲链表，用户环境（JOS 中的 process）链表等数据结构映射在相应的地址处。再往下的内存则存储一个进程运行时栈、堆、数据、代码等常见的内存块。

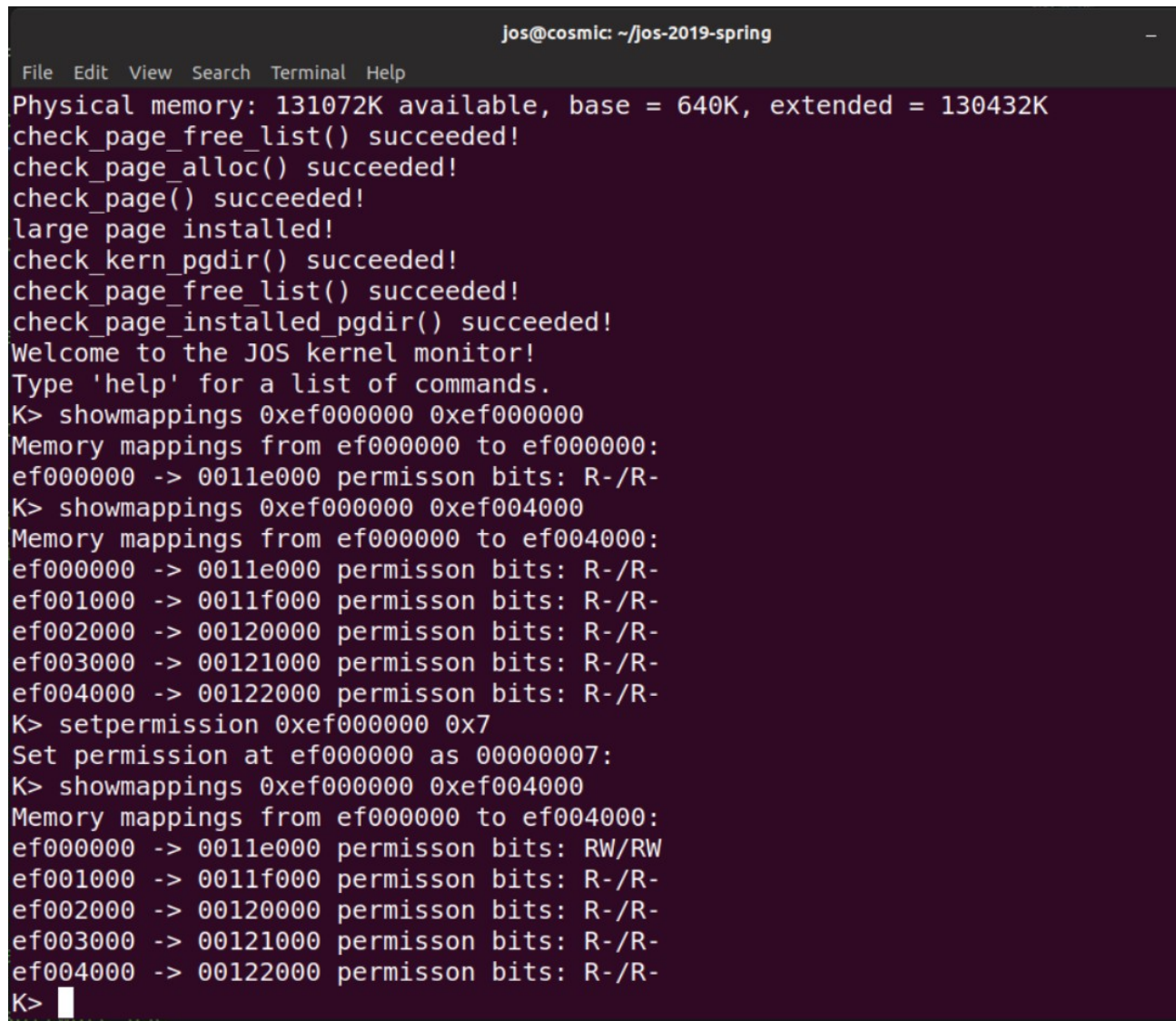
在实现 boot\_map\_region\_large 函数时，需要注意一个地方就是我们需要将 CR4 寄存器中的 page size 位置位，只有这样我们才可以使用大页，否则内核会发生错误无法运行。

## 五、实现一个 challenge

这次 lab 我选择实现了第二个 challenge，即在 monitor 上扩展命令来查看内存映射信息。我实现了三个指令：showmappings, setpermission, dumpmemory，分别提供的功能是查看虚拟内存映射，改变页表项

权限位，查看内存数据。在解析命令行时我使用了库提供的 `strtol` 函数来解析地址参数，之后使用在之前写好的 `pgdri_walk` 函数就可以找到相应的页表项，之后再根据功能需要访问更改页表项或者访问物理内存，这里需要注意的是在访问未被映射的虚拟地址以及未被分配的物理地址时，要注意查看页表项指针是否为空以及 `present` 位是否置位，及时打出错误信息避免出现内存错误。下面给出几张运行截图：

运行 `showmappings & setpermission`：



```
jos@cosmic: ~/jos-2019-spring
File Edit View Search Terminal Help
Physical memory: 131072K available, base = 640K, extended = 130432K
check_page_free_list() succeeded!
check_page_alloc() succeeded!
check_page() succeeded!
large page installed!
check_kern_pgdir() succeeded!
check_page_free_list() succeeded!
check_page_installed_pgdir() succeeded!
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K> showmappings 0xef000000 0xef000000
Memory mappings from ef000000 to ef000000:
ef000000 -> 0011e000 permisson bits: R-/R-
K> showmappings 0xef000000 0xef004000
Memory mappings from ef000000 to ef004000:
ef000000 -> 0011e000 permisson bits: R-/R-
ef001000 -> 0011f000 permisson bits: R-/R-
ef002000 -> 00120000 permisson bits: R-/R-
ef003000 -> 00121000 permisson bits: R-/R-
ef004000 -> 00122000 permisson bits: R-/R-
K> setpermission 0xef000000 0x7
Set permission at ef000000 as 00000007:
K> showmappings 0xef000000 0xef004000
Memory mappings from ef000000 to ef004000:
ef000000 -> 0011e000 permisson bits: RW/RW
ef001000 -> 0011f000 permisson bits: R-/R-
ef002000 -> 00120000 permisson bits: R-/R-
ef003000 -> 00121000 permisson bits: R-/R-
ef004000 -> 00122000 permisson bits: R-/R-
K> 
```

运行 dumpmemory:

```
josh@cosmic: ~/jos-2019-spring
File Edit View Search Terminal Help
Physical memory: 131072K available, base = 640K, extended = 130432K
check_page_free_list() succeeded!
check_page_alloc() succeeded!
check_page() succeeded!
large page installed!
check_kern_pgdir() succeeded!
check_page_free_list() succeeded!
check_page_installed_pgdir() succeeded!
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K> dumpmemory -v 0xef000000 0xef000100
Dump memory from ef000000 to ef000100:
0x00 0x00 0x00 0x00 0x01 0x00 0x00 0x00
0xf8 0xdf 0x15 0xf0 0x00 0x00 0x00 0x00
0x08 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x10 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x18 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x20 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x28 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x30 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x38 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x40 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x48 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x50 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x58 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x60 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x68 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x70 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x78 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
0x80 0xe0 0x11 0xf0 0x00 0x00 0x00 0x00
```