

**操作系统高级课程**

**XV6源码阅读报告**

**虚拟内存**

|  |  |
| --- | --- |
| **组长：** | 1901210759石江城 |
| **组员：** | 1901210765吴昊 |
|  | 1901210756梁梓豪 |
|  | 1901210360程人 |
|  | 1901210761王缤 |

**日期：2019年11月3日星期日**

* **阅读源码**

（1）首先是kalloc.c部分，其主要为用户空间、内核栈、页表及缓冲区分配物理内存。

1. kmem结构体定义了管理物理内存的空闲链表及相关锁。

2. kinit1()和kinit2()是初始化物理内存的两个函数，其使用时机不同，kinit1()在系统刚刚被加载进内存时使用，这时使用的页表是entrypgdir；在系统完成各种数据结构的初始化后，使用kinit2()将内存的剩余空间初始化。

3. freerange()给定的虚拟地址的起始和终止地址，释放这段空间对应的物理空间。kfree()接收虚拟地址，找到对应的物理地址，并释放以该地址为首址的一页空间，并将该页加入空闲链表。freerange()就是迭代使用kfree()来释放一段内存。

4. kalloc()从空闲链表中分配一页物理内存，返回对应的虚拟地址。

一页的大小PGSIZE为4096字节，即4K。

（2）vm.c实现虚拟内存功能，完成虚拟内存的初始化、分配和释放，包括GDT的创建等；进程切换时虚拟内存环境的切换，包括页目录的切换。

1. 虚拟内存初始化：

seginit建立内核段描述符

kmap结构体定义了内核页表kpgdir从虚拟内存空间到物理内存空间的映射。总的分为四个段，其中[0，0x1000000）主要存放bootloader和I/O设备；[0x1000000，data）存放内核代码；[data，0xE000000) 存放内核数据和其他可分配内存；[0xFE000000,0) 存放其他设备。

2. 内核虚拟内存：

kvmalloc分别调用setupkvm和switchkvm初始化内核页目录kpgdir，然后切换到内核虚拟空间。

setupkvm分配一页大小的内存存储内核页目录kpgdir，并利用kmap定义好的映射关系进行初始化。当系统没有进程时调度器就使用kpgdir来代替进程页目录；除了在main()中调用该函数以外，每个用户程序创建时，也通过setupkvm获取包含内核程序映射的页表pgdir，然后通过allocuvm为用户程序分配内存，loaduvm将用户程序载入内存并添加映射。因此，每个进程的虚拟地址空间都包含内核程序和用户程序。

switchkvm设置cr3寄存器的值为kpgdir首地址，切换entrypgdir到kpgdir；

3. 用户虚拟内存：

inituvm分配一页物理页，将页表中初始项映射到该内存，把用户程序的初始化代码加载到此处；allocuvm增大用户虚拟空间大小，用于为分配给用户程序；loaduvm加载用户程序段到内存，并添加映射到页目录；deallocuvm减小用户虚拟空间大小；copyuvm复制指定的用户进程空间；

switchuvm用户进程切换时，负责相关数据结构的载入等操作，设置好该用户进程需要的虚拟内存环境，包括通过设置cr3将系统页表切换到用户进程的页表。

4. 常用工具类：

walkpgdir给定虚拟地址，返回该地址对应的二级页表中的页表项。如果虚拟地址对应的页表还没被创建，则可以建立并对其进行初始化；

mappages给定虚拟地址空间va-va+size，将这段虚拟空间在页表里创建物理地址映射。该函数调用walkpgdir找到虚拟地址对应的二级页表项，然后为其填充物理地址pa，权限perm和有效位P。

freevm释放指定页目录记录的所有物理页框；clearpteu清除页表中的指定页表项；uva2ka配合copyout使用，映射用户虚拟地址到内核地址；copyout从给定的物理地址处复制len字节到给定的用户地址。

* **回答问题**

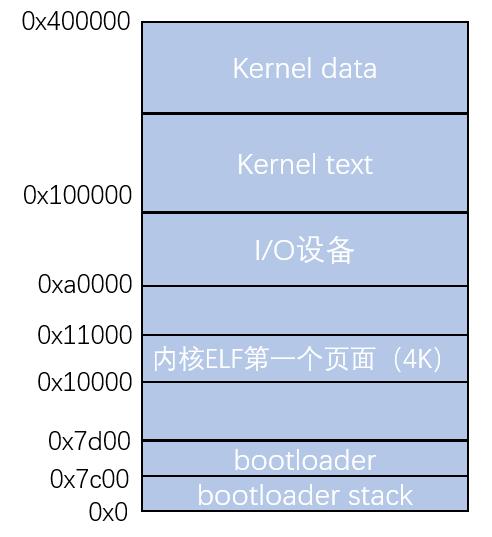
1. **XV6 初始化之后到执行 main.c 时，内存布局是怎样的（其中已有哪些内容）？**

BIOS初始化后，将xv6的bootloader(包括bootmain.c和bootasm.S)加载到物理内存，然后将控制权交给bootloader。bootmain()将内核的ELF文件第一个页面从硬盘加载进内存0x10000处，接下来加载其他程序段到0x100000，最后是调用内核的入口指令，也就是entry.s中的entry。

entry.S的主要任务是设置页表entrypgdir，让分页硬件能够正常运行。因为在初始化整个物理内存的页表kpgdir和空闲链表之前，本身就需要分配一块放置kpgdir的内存。而分配内存的分页硬件本身又需要一个页表，这个页表就是entrypgdir。entrypgdir将虚拟地址[0, 4MB)和[KERNBASE, KERNBASE+4MB)都映射到了物理地址空间[0, 4MB)。由于此时部分代码还运行在虚拟地址低地址处，因为必须将[0, 4MB)映射到[0, 4MB)。而直到main()里执行kinit1()切换页表为kpgdir之前，都依靠entrypgdir将其余代码的虚拟地址[KERNBASE, KERNBASE+4MB)转换到[0, 4MB)。

entry.S最后跳转到main.c的main()函数处，开始整个操作系统的运行。

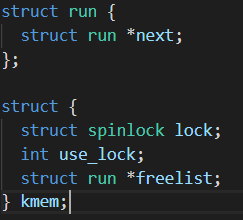
所以在执行main.c前，内存中包括bootloader，已分配I/O空间，内核指令和数据，包括初始页目录entrypgdir。内存中的内容如下图：



1. **XV6 的动态内存管理是如何完成的？ 有一个 kmem（链表），用于管理可分配的物理内存页。（vend=0x00400000，也就是可分 配的内存页最大为 4Mb）？**

物理内存空间上空闲的每一页，都有一个指针域（虚拟地址）指向下一个空闲页，最后一个空闲页为NULL ，通过这种方式，kmem只需要保存着虚拟地址空间上的freelist地址即可。

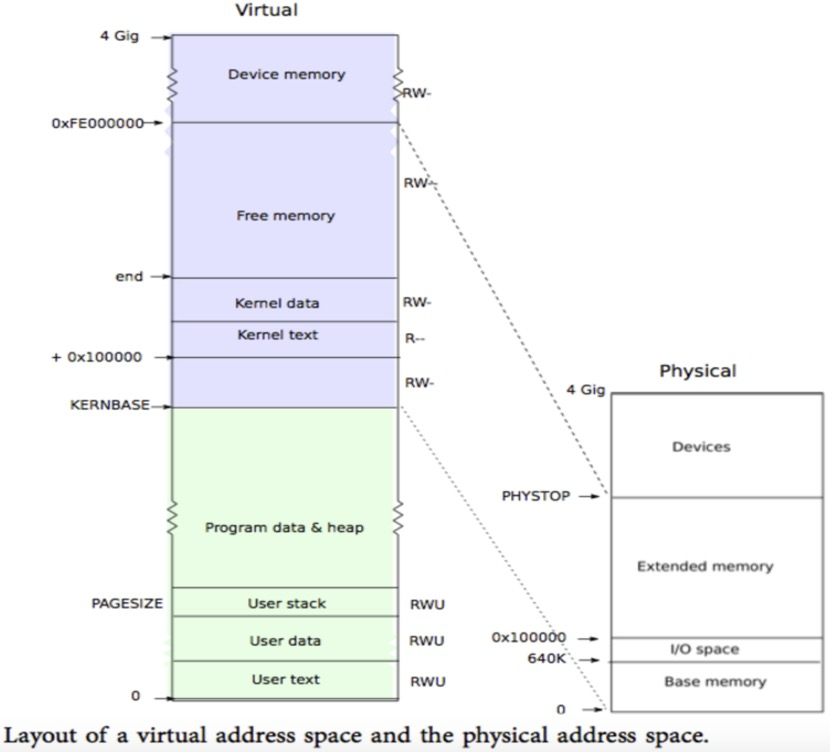
对freelist的维护在函数kfree和kalloc里实现。kfree接收一个虚拟地址v，用填充1的方式抹除其数据，然后作为一个空闲页节点添加到freelist的链表头。kalloc则返回freelist的链表头。



1. **XV6 的虚拟内存是如何初始化的？画出 XV6 的虚拟内存布局图，请说出每一部分对应的内容是什么。 见 memlayout.h 和 vm.c 的 kmap 上的注释？**

在执行main.c之前，内核将ELF文件加载到内存后的第一条地址end。因为这时使用的页表仍是最初的页表entrypgdir，只能初始化4M（210\*212），xv6通过在main函数最开始处释放内核末尾到4M的空间来分配页表kinit1(end, P2V(4\*1024\*1024))。

接着kvmalloc()在内核构建了内核页表kpgdir后，能够完全访问内核的虚拟地址空间，在各种表目、缓存等数据结构初始化之后，初始化其余物理内存（仅仅到PHYSTOP）， kinit2(P2V(4\*1024\*1024), P2V(PHYSTOP)).



具体的映射：

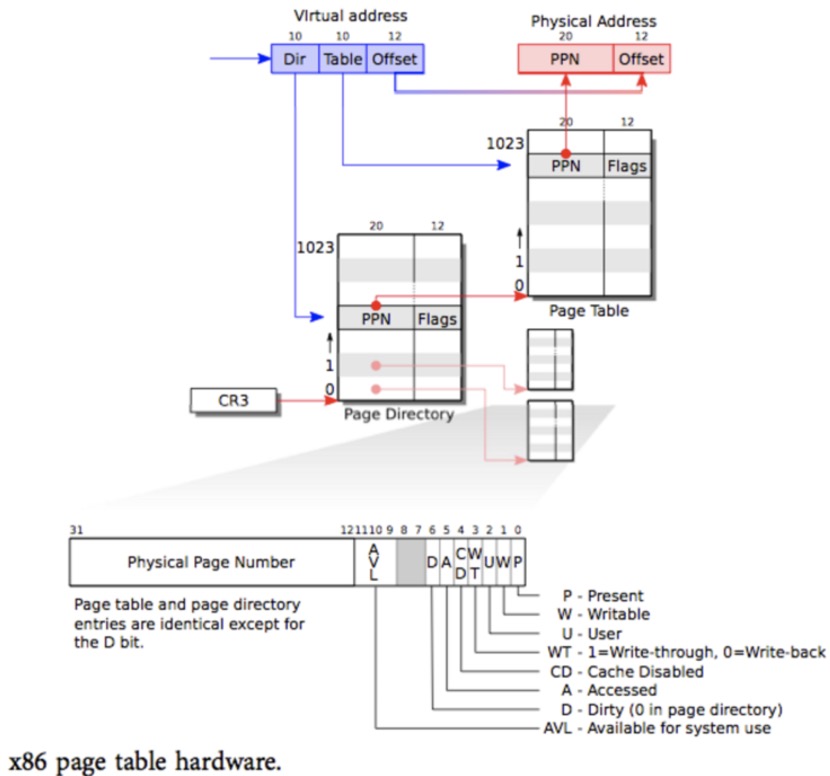
* 0——KERNBASE→?：这部分为用户内存空间，由内核动态分配物理内存，无确定映射关系。
* KERNBASE——KERNBASE+EXTMEM→0——EXTMEM：用于I/O设备空间、bootloader、内核初始化代码等（EXTMEM=0x100000）
* KERNBASE+EXTMEM ­——data→EXTMEM——V2P(data)：data是bootmain()将内核的ELF文件加载进内存后的end地址，这部分主要是内核指令和数据。
* data——KERNBASE+PHYSTOP→V2P(data)——PHYSTOP：自由分配的物理内存，包括内核栈、管道缓冲区、用户内存与页表。（PHYSTOP=0xE000000）
* KERNBASE+PHYSTOP——KERNBASE +0xffffffff→PHYSTOP——0xffffffff：其它设备空间，例如高级可编程中断控制器（ioapic）。

1. **关于XV6 的内存页式管理。发生中断时，用哪个页表? 一个内存页是多大? 页目录有多少项? 页表有多少项? 最大支持多大的内存? 画出从虚拟地址到物理地址的转换图。在XV6中，是如何将虚拟地址与物理地址映射的(调用了哪些函数实现了哪些功能)？**

发生中断时，将获得CPU的进程的页目录（一级页表）首地址的从该进程的PCB复制到CR3寄存器。

一个内存页的大小为212，即4k；32位地址中，低12位为页内偏移地址。高10位为页目录索引，中间10位为页表索引，所以页目录有210项，每个页表各有210项。由于XV6采用32位地址，则最大支持232，即4G空间。

虚拟地址和物理地址的转换函数V2P和P2V就是加减一个KERNBASE。所以物理地址中的0——4G对应虚拟地址中的KERNBASE——KERNBASE+4G。



* **参考文献**

[1]鲜染.赵天雨.胡树伟.胡文涛.曹扬.XV6中文文档[EB/OL]. https://th0ar.gitbooks.io/xv6-chinese/content/index.html

[2] 佛无欲. 如何保存CPU现场信息[EB/OL]. https://blog.csdn.net/u011454830/article/details/49704825, 2015-11-07.

[3] Bin\_LmmH\_C. PEB结构学习[EB/OL]. <https://www.cnblogs.com/binlmmhc/p/6501545.html>, 2017-03-04