# 第 2 章 虚存管理

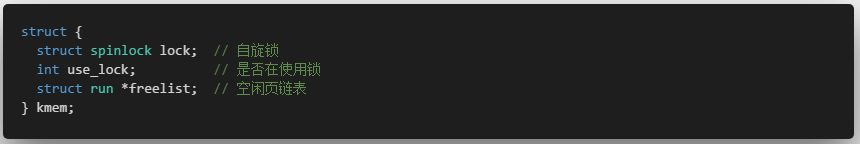
## 源码阅读

**阅读要求：** 内存管理部分： kalloc.c vm.c 以及相关其他文件代码

**源码阅读：**

虚拟内存是指将内存与外存有机的结合起来使用，从而得到一个容量很大的“内存”。这是一种以时间空间（外）换取空间（内）的技术。虚存管理，是现代操作系统不可缺失的一种管理机制。

在kalloc.c文件中，定义了物理内存分配器kmem。其核心数据为空闲页链表，通过对物理空闲页的分配和回收来管理物理内存，其数据结构如下：

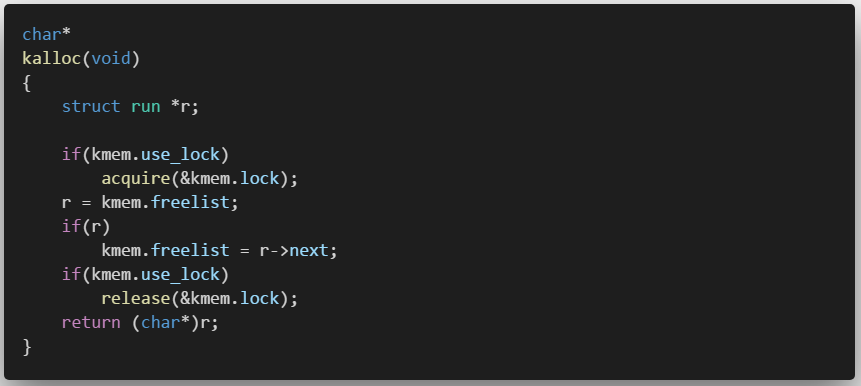


在该文件中也有分配器的初始化函数，空闲页的分配和回收函数。其中，变量lock的意义为：当lock为1时代表锁已被占用，反之未被占用，初始值为0。

同时，在调用锁之前，必须对锁进行初始化。其初始化方法分两个阶段进行。首先，main()主函数调用kinit1()方法，同时仍然使用entrypgdir来放置just。由entrypgdir映射到空闲列表的页面。 最后，main()主函数使用其余物理页面调用kinit2()方法，在安装了一个完整的页面表之后，将它们映射到所有核心上。

kalloc ()方法和kfree ()方法分别实现对空闲页的分配和回收的功能。

kalloc ()方法：首先，使用acquire()方法（该方法在同步机制中介绍）申请分配器的lock；然后，获取空闲页链表的链首，并将空闲页链表指针后移；最后，使用release()（该方法在同步机制中介绍）释放分配器的lock。



kfree ()方法：如同分配一样。首先，使用acquire()方法申请分配器的lock；然后，获取空闲页链表的链首，并将释放的页面挂在空闲页链表的链首；最后，使用release()释放分配器的lock。

每个进程有一个页面表，加上CPU不运行任何进程时使用的页面表(kpgdir)。内核在系统调用和中断期间使用当前进程的页表;页面保护位阻止用户代码使用内核的映射。vm.c文件主要功能就是提供维护这些页表的方法。本文将在下面的讨论区更多的介绍这一块

## 讨论总结

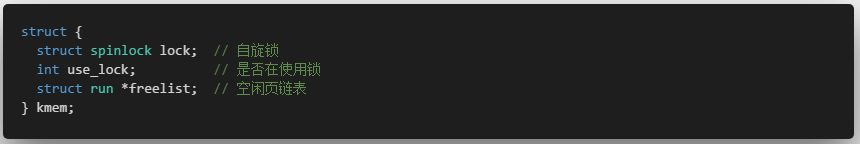
**XV6 初始化之后到执行 main.c 时，内存布局是怎样的（其中已有哪些内容）？**

当XV6 初始化之后，执行到main()方法之前，物理地址的具体内容如下。其中，引导程序是由BIOS负责载入内存，而设备区则是硬件规定占用的区域，内核ELF文件头和Xv6操作系统是由引导程序(bootmain.c)加载进内存的。



**XV6的动态内存管理是如何完成的？**

正如上一节所述，在kalloc.c文件中，系统定义了一个物理内存管理模块kmem。其核心数据为空闲页链表，通过对物理空闲页的分配和回收来管理物理内存，其数据结构如下：



在该文件中也有分配器的初始化函数，空闲页的分配和回收函数。其中，变量lock的意义为：当lock为1时代表锁已被占用，反之未被占用，初始值为0。

同时，在调用锁之前，必须对锁进行初始化。其初始化方法分两个阶段进行。首先，main()主函数调用kinit1()方法，同时仍然使用entrypgdir来放置just。由entrypgdir映射到空闲列表的页面。 最后，main()主函数使用其余物理页面调用kinit2()方法，在安装了一个完整的页面表之后，将它们映射到所有核心上。

**XV6 的虚拟内存是如何初始化的？**

在XV6操作系统中，通过使用end指针来标记Xv6的ELF文件的结尾地址，从而使得[PGROUNDUP(end), 0x400000]范围内的物理页可以被用作内存页来进行分配。正如上一个问题中介绍的，main()主函数调用kinit1(end, P2V(0x400000))将这部分内存纳入虚拟内存页管理。

kmem内存分配器必须知道可用的空闲页的内存范围。由于此时虚拟内存已经开启，且页表表项只有两条，因此Xv6必须利用已有的虚拟地址空间，在其中创建新的页表。kinit1()函数会调用freerange()函数，按照前文叙述的方式，建立从PGROUNDUP(end)地址开始直到0x400000为止的全部内存页的链表。kvmalloc()函数获得一个虚拟内存页并将其初始化一级页表。

最后，main()主函数使用其余物理页面调用kinit2()方法将[0x400000, 0xE00000]范围内的物理地址纳入到内存页管理之中，在安装了一个完整的页面表之后，将它们映射到所有核心上。

**XV6 的虚拟内存布局图，请说出每一部分对应的内容是什么。**

Xv6操作系统的虚拟内存布局图及其对应的每一部分内容如下表所示。正如我们所了解的，每个用户进程都有一个页表。其中，有关内核部分(即最后四项)，对于所有用户进程都是一样的。而前面的映射会有所不同，表中的信息根据init的进程的ELF文件信息和exec调用的代码确定。



**发生中断时，用哪个页表？**

中断发生时，使用的的页表依然是对应用户进程的页表。由于用户地址空间不仅有用户栈还有内核栈，因此陷入的内核代码依然可以正常执行。只有当中断处理程序决定退出当前进程或者切换到其他进程时，当前页表才会被切换为调度器的页表(全局变量kpgdir)，并在调度器中切换为新进程的页表。

**一个内页是多大？页目录有多少项？页表有多少项？最大支持多大的内存？**

XV6操作系统采用二级页表来管理虚拟内存。在这个二级页表结构中，每个页的大小为4KB，每个页表的大小也为4KB，每个页表项的大小为4字节，一个页表包含1024个页表项。一级页表表项存储的是二级页表的地址，二级页表表项存储的是对应的物理地址。虚拟地址和物理地址的最后12位总是相同，因此页表表项中的这12位可以被用作标记其他信息。

**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）？**

userinit()方法的主要功能是来初始化用户进程。userinit()在完成有关进程数据结构管理的工作后，会初始化这个进程自己的页表(struct proc中的pgdir)。而将虚拟地址与物理地址映射的正是在这一步进行。

首先，userinit()会使用setupkvm()生成与前述一模一样的内核页表，然后使用inituvm()生成第一个用户内存页(映射到虚拟地址0x0)，并将用户进程初始化代码移动至这个内存页中。

initcode.S中包含了一个exec系统调用，通过这个系统调用来加载进一个真正的用户进程。exec会从磁盘里加载一个ELF文件。ELF文件中包含了所有代码段和数据段的信息，并且描述了这些段应该被加载到的虚拟地址(这是在编译时就已经确定好的，所以编译器必须遵循某些约定来分配这些虚拟地址)。

最后，exec会分配两个虚拟内存页，第一个页设置为不可访问，第二个页用作用户栈。