《操作系统》

课程讲稿

第三章 第3讲

核心数据结构和算法

软件所制

第三章 第3讲 核心数据结构和算法

**学时：**1学时

**教学目的：**学习内存数据结构，理解页和区的关系与区别。学习内存分配函数以及对应的算法。

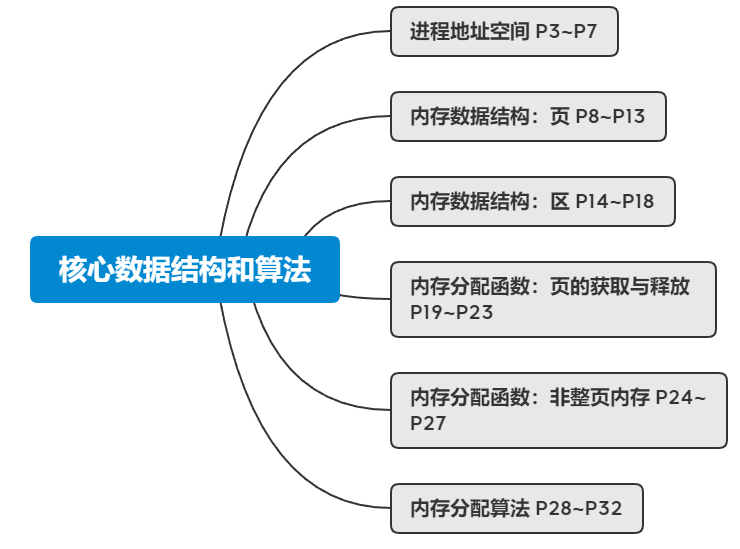
**课程时间线：**



**课外参考读物：**

《操作系统导论》

**知识框图：**

****

PPT讲稿：

2. 这张图给出了逻辑地址到物理地址的转换过程，逻辑地址通过分段机制转换为线性地址，再通过分页机制转换为物理地址。
3. 上图的各个阶段，包括了分段机制：即分成代码段，数据段，堆栈段。每个内存段都与一个特权级相关联，即0~3，0具有最高特权级（内核），3则是最低特权级（用户）。每个段都有一个段选择符。段描述符指明段的大小、访问权限和段的特权级、段类型以及段的第一个字节在线性地址空间中的位置（称为段的基地址）。而段选择符用于在描述符表中进行索引找到段描述符。虚拟地址的偏移量部分加上段的基地址上就可以定位段中某个字节的位置，即形成线性地址空间中的地址。当使用分页机制时，每个段被划分成页面（通常每页在4KB大小），页面会被存储于物理内存或硬盘上。如果禁用分页机制，那么线性地址空间就是物理地址空间。
4. 这张图给出了32位系统下的进程地址空间，每个进程有4G的地址空间，其中0G-3G为用户空间，3G-4G为内核空间。
5. 而到了64位系统下，理论上有2^64(16EB)大小的虚拟地址空间，实际上是2^48(256TB)大小的虚拟空间。因为只用到了其中48位进行寻址。其中低128T为用户空间，高128T为内核空间。
6. 我们介绍一下内存管理中主要的数据结构，页。
7. 内核把物理页作为内存管理的基本单位。体系结构不同，支持页的大小也不同。一般32位的体系结构支持4KB大小的页，64位的体系结构支持8KB大小的页。Linux下查看页大小可以通过getconf PAGESIZE
8. 页的数据结构如图所示，包括了标签，引用计数等信息。
9. 其中，flags是一组描述页状态的标志。例如：PG\_locke是页被锁定。例如，在磁盘I/O操作中涉及的页。PG\_error 在传输页时发生I/O错误。PG\_referenced 刚刚访问过的页。PG\_uptodate 在完成读操作后置位，除非发生 磁盘I/O错误。PG\_dirty表示页已经被修改。PG\_lru表示页在活动或非活动页链表中
10. \_count变量为页的引用计数：如果字段为-1，表示相应页空闲，并可被分配给任一进程或者内核本身；如果字段值大于等于0，则说明页被分配给了一个或多个进程，或者用于存放一些内核数据结构。使用page\_count()函数可以返回\_count加1后的值，也就是该页的使用者的数目。\*virtual指针指向页的虚拟地址，通常情况下，它就是页在虚拟内存中的地址。
11. 系统中的每一个物理页都要分配一个这样的结构体：struct page占40字节内存，而一个物理页的大小为4KB。如果系统有4GB物理内存，那么系统中共有2^20个页，用于描述页面的page结构体消耗的内存为40MB，因此代价并不算太高
12. 我们再来介绍内存中的另一个重要数据结构Zone，区
13. 啊由于硬件的限制，有些页位于内存中特定的物理地址上，所以不能将其用于一些特定任务。比如一些硬件只能用某些特定的内存地址来执行DMA；一些体系结构的内存的物理寻址范围比虚拟范围大得多，就有一些内存不能永久地映射到内核空间上。内核把页划分为不同的区，内核使用区对具有相似特性的页进行分组。
14. Linux主要使用了三种区。ZONE\_DMA这个区包含的页能用来执行DMA操作；ZONE\_NORMAL这个区包含的都是能正常映射的页；ZONE\_HIGHEM这个区包含“高端内存”，其中的页不能永久地映射到内核地址空间。
15. Zone的数据结构如图所示。区的结构体很大，但是系统中只有3个区
16. 从特定的区中获取页,用于DMA的内存必须从ZONE\_DMA中分配;从多个区中获取页,一般用途的内存既能从ZONE\_DMA中分配，也能从ZONE\_NORMAL分配;不能同时从两个区中分配,分配不能跨区界限.如果可供分配的资源不够，内核会占用其他可用区的内存
17. 接下来我们介绍一下页的获取与释放
18. 这是操作系统页相关的API函数，包括页的分配，释放，其中包括分配单个或连续个页，也可以同时释放单个或连续个页。
20. 要注意的是，释放页时只能释放属于自己的页，传递了错误的struct page地址，用了错误的order值都会导致系统崩溃。
21. 这是一个获取和释放页面的例子，我们看到参数是3，代表需要获取连续8个页，如果没有获取成功，一般是没有足够空闲内存，并返回错误码。
22. 接下来我们介绍内存分配函数
23. 最典型的内存分配函数是kmalloc，这是一个按字节获取内存的函数调用。函数原型是void \*kmalloc(size\_t size, int flags)。Flags为GFP\_KERNEL、GFP\_ATOMIC等，可以在(linux/gfp.h)这个文件里获取。他的头文件是#include <linux/slab.h>。函数调用成功，返回虚拟地址，分配的内存在物理上是连续的。最多只能分配32\*PAGESIZE大小的内存，最小处理32字节或者64字节的内存块。分配速度较快，是内核中主要的内存分配方法。释放时直接调用kfree(void \*ptr)
24. Vmalloc同样是分配内存的函数。其函数原型是void \*vmalloc(unsigned long size)。size以字节为单位。头文件是#include <linux/vmalloc.h>。其虚拟地址连续，物理地址不连续，最小可处理4KB的内存块，分配速度较慢，一般用于大块内存的获取。释放直接调用vfree()
25. 这里给出了几个典型的内存分配函数，包括kmalloc()和vmalloc()是分配的是内核的内存。malloc()分配的是用户的内存。realloc()用于调整已分配的用户内存的大小。
26. 接下来我们介绍典型的内存分配算法
27. 固定分区和动态分区方式都有不足之处；固定分区方式限制了活动进程的数目，当进程大小与空闲分区大小不匹配时，内存空间利用率很低；动态分区方式算法复杂，回收空闲分区时需要进行分区合并等，系统开销较大；伙伴系统方式是对以上两种内存方式的一种折衷方案；
28. 伙伴系统的实现过程是把所有空闲页分组为11个块链表，每个块链表分别包含大小为1，2，4，8，16……1024个连续页的页块。其中最大可申请1024个连续页（即4MB连续内存），每次申请时寻找最小满足要求的块链表，找到后分配内存并将剩余内存链接到较小的链表上。找不到就去更大的块链表寻找。
29. 例如，申请一个256页大小的块，先从256页的链表中查找。如果发现256页链表中没有空闲块，去512页的链表中查找。发现512页链表中没有空闲块，去1024页链表查找。在1024页链表中找到空闲块，则将其中的256页作为请求返回，剩余的768分成256块和512块分别插到相应的链表中。如果在1024页链表中仍然没有找到，则返回错误。
30. 页块释放时，会被加到相应的页块链表后面。如果两个页块大小一样，且物理地址连续，则合并成一个页块。（这两个页称为伙伴页）。伙伴页合并是一个迭代的过程。