《openEuler内核编程》

课程讲稿

第三章 第4讲

内存操作常见问题

软件所制

第三章 第4讲 内存操作常见问题

**学时：**1学时

**教学目的：了解内存需要分配的原因，学习不同种类的内存分配方法。掌握几种经典的内存置换算法。了解内存映射机制可以带来的性能优化。**

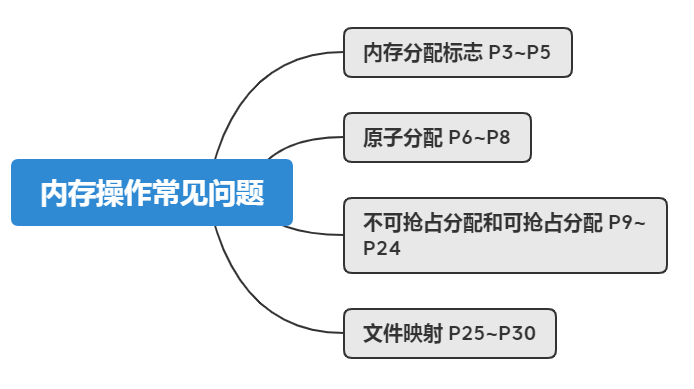
**课程时间线：**



**课外参考读物：**

《操作系统导论》

**知识框图：**

****

PPT讲稿：

3. GFP\_ATOMIC这个标志用在中断处理程序、持有自旋锁以及其他不能睡眠的地方。GFP\_NOWAIT与GFP\_ATOMIC类似，不同之处在于，调用不会退给紧急内存池。这就增加了内存分配失败的可能性。GFP\_NOIO这种分配可以阻塞，但不会启动磁盘I/O。这个标志在不能引发更多磁盘I/O时阻塞I/O代码，这可能导致令人不愉快的递归。GFP\_NOFS这种分配在必要时可能阻塞，但不会启动文件系统操作。这个标志在你不能再启动另一个文件系统的操作时，用在文件系统部分的代码中。GFP\_KERNEL是一种常规分配方式，可能会阻塞。这个标志在睡眠安全时用在进程上下文代码中。为了获得调用者所需的内存，内核会尽力而为，这个标志应当是首选标志。GFP\_USER是一种常规分配方式，可能会阻塞。这个标志用于为用户空间进程分配内存。GFP\_HIGHUSER是从ZONE\_HIGHMEM进行分配，可能会阻塞。这个标志用于为用户空间进程分配内存。GFP\_DMA是从ZONE\_DMA进行分配，需要获取能供DMA使用的内存的设备驱动程序使用这个标志，通常与以上某个标志组合在一起使用。
4. 不同情况下应使用的标志。进程上下文，可以睡眠使用GFP\_KERNEL。进程上下文，不可以睡眠；中断处理程序；软中断和tasklet使用GFP\_ATOMIC。需要用于DMA的内存，可以睡眠使用（GFP\_DMA | GFP\_KERNEL）。需要用于DMA的内存，不可以睡眠使用（GFP\_DMA | GFP\_ATOMIC）
5. 接下来我们介绍一下原子分配
6. 普通分配（GFP\_KERNEL）获取新的页面时可能需要换出其他页，因为对换需要时间，进程会等待它完成，这时内核可以调度执行其他任务。原子分配（GFP\_ATOMIC）由于内核并不允许通过换出数据或缩减文件系统缓冲区来满足原子分配请求，所以必须还有一些真正可以获得的空闲内存。
7. 为了尽量保证原子请求不失败，内核为原子内存分配保留了一个页框池，只有在内存不足时使用。但无法保证原子分配不失败。
8. 接下来介绍一下抢占和不可抢占的分配方式。
9. 静态内存分配的内存地址和大小在程序加载和执行就已经确定，且无法改变。动态内存分配氛围两种方式，不可抢占分配包括首次适应、循环首次适应、最佳适应、最差适应等分配算法。可抢占分配则希望重新分配正在使用的内存，已获得更大的连续内存空间。
10. 首次适应算法每次从低地址开始查找，直到找到满足大小要求的空闲分区。优点是保留高地址空间的大空闲区。缺点是低地址区不断被划分，留下许多碎片，且查找开销大。循环首次适应算法是每次从上次适配的地址开始寻找，其优点为内存分区分布得更加均匀，缺点则是缺乏大的分区空间
11. 最佳适应算法是把空闲区按大小排序，每次寻找最适合的内存区。其优点是对于每次分配是最优的，浪费的空间小。缺点是会留下好多小的空闲区；排序会产生额外的开销。最差适应算法是把空闲区按大小排序，每次从最大的空闲区分配。其优点是分配时不需要和空闲区逐个比较；碎片较少。缺点是缺乏大的空闲区；排序会产生额外的开销
12. 这是利用最佳、最差和首次适应算法进行内存分配的例子。我们可以看到新任务被分配到了不同的内存地址空间，同样造成了不同大小的碎片。
13. 可抢占分配包括压缩和替换两个主要过程。压缩是已经存在的内存块重新分配地址，从而为新的程序获得一个更大的空闲空间。替换是将一些已经分配的空间可以被再次使用（需要一种规则来寻找这些空间）。同时需要区分dirty块和clean块的区别，覆盖clean块时，直接重写即可；覆盖dirty块时，需要在另一处内存备份。Dirty块：多次被改写，往往是程序的数据部分。Clean块：没有被改写，往往是程序的指令部分。
14. 如图所示，压缩后能大大增加连续的空闲空间。但是压缩的速度非常慢
15. 内存置换的评价标准是引用内存块是命中率尽可能高（或者说是最少的缺页）
16. FIFO方法在内存有空闲时依次装入需要分配的页面。按照页面装进内存的时间进行置换，越老的页面最先被换出，不管该页面是否经常使用。该方法有可能导致缺页率增加，导致页面置换次数增加。
17. 这是一个典型的FIFO算法的例子。我们看到有\*号的是最老的页面，因此，每当内存满的时候，有\*号的页面会被优先患换出。
18. LRU方法在内存有空闲时依次装入需要分配的页面。按照上次使用时间进行排序，将离上次使用时间最长的页面置换出。可以采用栈的数据结构，每次页面被访问将该页面号放在栈顶
19. 这是LRU算法的示例，我们可以看到，其实就是一种栈结构的数据使用。
20. OPT方法在内存有空闲时依次装入需要分配的页面。将不再使用的页面换出，而实际中不能预知哪个页面不再使用，因此会把之后要装入内存的页号查一遍，计算出最晚再次会被装入的页号。这个算法是最优算法，可以作为评测其他算法的性能
21. 这是实例，大家自己对照算法进行内存的填入。可以看到其hit率比前两种算法高很多。
22. 其他方法还有最近未使用页面置换算法，即设置引用位R，每次调用将R=1，系统每隔一段时间将R=0，当进行置换式检查哪个页面为零说明近期不会再使用，可以将其换出。时钟页面置换算法，即设置引用位R，当R=1说明被引用过，这种方法是FIFO的改进，根据装入内存时间和是否被引用过作为标准，首先从时间最长项检查，若R=0则置换出，若为1则检查下一项并将R=0。若全部R=1，则按照FIFO方法进行置换。
23. 这是所有内存分配相关算法的导图，大家可以清晰的分辨不同类型的分配方式和其相关算法
25. 内存映射文件与虚拟内存有些类似。映射文件到内存后，可以直接操作这段虚拟地址进行文件的读写操作，不必再调用read/write等系统调用。
26. 传统的文件访问，需要通过页表，再通过缓存，最后进入磁盘访问文件。而通过映射方式，每一个进程在自己的地址空间都包含有该文件的副本，这不必要地浪费了存储空间。当两个进程同时读一个文件的同一页时，系统要将该页从磁盘读到高速缓冲区中，每个进程再执行一个存储器内的复制操作将数据从高速缓冲区读到自己的地址空间。
27. 而共享储存映射是指进程A和进程B都将该页映射到自己的地址空间，当进程A第一次访问该页中的数据时， 内核读入这一页到内存并更新页表使之指向它。以后, 当进程B访问同一页面时, 该页已经在内存, 内核只需要将进程B的页表登记项指向此页即可。
28. 内存映射取消了将文件数据加载到内存、数据从内存到文件的回写以及释放内存块等步骤，使得内存映射文件在处理大数据量的文件时能起到相当重要的作用。实际工程中的系统往往需要在多个进程之间共享数据，内存映射文件可以使进程之间通过映射同一个普通文件实现共享内存。
29. 内存映射步骤，第一步、在用户虚拟地址空间中寻找空闲的满足要求的一段连续的虚拟地址空间，为映射做准备。通过mmap系统调用准备这样一段虚存空间,并建立一个结构体,将其传给具体的设备驱动程序。第二步、建立虚拟地址空间和文件或设备的物理地址之间的映射。第三步、当实际访问新映射的页面时的操作。由缺页中断完成。