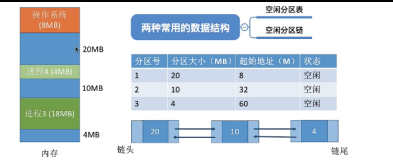


动态分区算法

首次适应
(效果最好)

- 思想 —— 每次从低地址查找，找到一个能满足大小的空闲分区
- 实现 —— 空闲分区以地址递增的次序排列。
每次分配内存时按顺序查找空闲分区链，找到大小满足要求的第一个空闲分区
- 保留住了高地址的大分区 —— 最佳适应算法的优点



最佳适应

- 思想 —— 尽可能留下大片的空闲区，为大进程服务
优先使用空间更小的分区
- 实现 —— 空闲分区按照容量递增的次序链接
每次分配内存按顺序查找空闲分区表
找到大小能满足要求的第一个空闲分区
- 每次分配完需要对空闲分区链进行重新排序
- 缺点 —— 每次选择最小的分区分配。
小的难以利用的内存块越来越多
外部碎片很多

最坏适应

- 思想 —— 优先使用最大的连续空闲区，剩下的空闲区不会太小，更方便利用
- 实现 —— 空闲分区按照容量递减的次序链接
每次分配内存按顺序查找空闲分区表
找到大小能满足要求的第一个空闲分区
- 每次分配完需要对空闲分区链进行重新排序
- 缺点 —— 导致连续的空闲取被迅速用完。如果之后有大进程达到，没有内存分区可以使用

邻近适应算法

- 思想 —— 首次自适应时，每次都从低地址开始查找，导致低地址处出现了很多小的空闲分区，而每次分配查找时，都要经过这些分区，增加了查找的开销
每次从上次查找结束的位置开始检索，解决上面问题
- 实现 —— 按照地址递增的顺序排列成为循环链表
每次从上次查找的结束位置开始查找空闲分区链
找到大小满足要求的第一个空闲分区
- 每次分配完后，不需要重新排列
开销更小
- 高地址部分的大分区更有肯能被用完，导致无法分区可以使用 —— 最大适应的缺点

框架

知识回顾与重要考点				
算法	查找策略	分区排列顺序	优点	缺点
首次适应	从头到尾找符合条件的分区	空闲分区以地址递增次序排列	包含碎片能最好， 碎片最少，空闲分区数一般不需要对空闲分区表重新排序	
最佳适应	优先使用最小的分区，以保留更多大分区	空闲分区以容量递增次序排列	会有更多的大分区被保留下来，更能满足大进程需求	会产生很多太小的、难以利用的碎片； 每次分配大、小块分区后都要对空闲分区表重新排序
最坏适应	优先使用更大的分区，以防止产生太多难以利用的碎片	空闲分区以容量递减次序排列	可以减少难以利用的小碎片	大分区容易被用完，不利于大进程； 算法开销大（排序上）
邻近适应	由首次适应演变而来，每次从上次查找结束位置开始查找	空闲分区以地址递增次序排列，可得到成循环链表	不用每次都从低地址的小分区开始检索， 算法开销小（使用回首次适应算法）	会使高地址的大分区表被用完

