连续分配管理方式

内存划分与内存分配算法

不同的划分，对应不同放入分配算法。

|  |  |  |  |
| --- | --- | --- | --- |
| 固  定  分  区 | 各分区大小相等 | 进程太大装不下，进程太小独占一区浪费 | 任选一空分区 |
| 适合多个功能完全一样的进程运行 |
| 各分区大小不等 |  | 增序排列，挑选最小但容得下进程的空分区 |
| 装的下大进程，小进程浪费减少 |

固定分区有n个分区，能运行的进程的数目也是n，所以固定分区的缺点是运行进程数目受到限制不灵活，且仍会浪费空间。

动态划分

事先不分区，根据进程申请内存的大小动态的建立分区。初始可以把用户内存看做一个分区，并用数据结构记录分区情况。

分配算法：

1. Frist Fit 把空闲分区按地址递增次序链成一个链表，从低地址往高地址找，找到的第一块容得下进程的空分区，把多余的切下留在链表里，剩下的分出去。

优点：每一次都是从低地址开始找和切割，保留了高地址的大分区，有利于装下后续可能出现的大进程。

缺点：不断切割低地址，使低地址出现很多无法利用的“碎片”。

1. Next Fit 在Frist Fit的基础上，从上次切割的分区后面开始找，循环……

优点：使适宜进程的大小的空分区分布均匀，减少了查找适宜空分区的时间。

缺点：导致缺乏大分区，后续无法装入大进程。

1. Best Fit 将空闲分区按容量递增排列，从头找到最小的但是能装下进程的空闲分区。

优点：每次切割的都是较小的分区，不动大分区，有利于大进程。

缺点：造成每次切割下的碎片都是最小的，更加难以利用，碎片会越来越多。

1. Worst Fit将空闲分区按容量递减排列，从头找到最大的能装下进程的空闲分区。

优点：每次切割的留下的分区，都尽可能大，再被利用的可能性增加，碎片少了。

缺点：没有大分区。

回收策略：相邻合并，不相邻创建数据项，记录此空闲分区大小，首地址等信息。

固定分区中没被利用的空间叫内部碎片。

动态分区中检测到太小装不下的空间叫外部碎片。

伙伴系统

所有分区都是2的k次幂，一个进程大小是n，2^(i-1)<n<2^i，那么找到一个大小是2^i的空分区给进程，如果不存在2^i的，就把2^(i+1)拆成两个2^i，把其中一个给进程,如果2^(i+1)的也没有，就找2^(i+2)的拆成4份，把其中一份给进程……

回收时，把地址紧邻且大小相等的空分区合并，不符合此条件的不合并。

链接装入和重定位

程序与主存

1. 编译程序编译源程序生成若干目标模块

目标模块的起始地址都是0号单元

1. 链接程序把所有目标模块与库函数链接成装入模块

拼接[模块的地址]使之统一从0号单元编址

1. 装入程序把装入模块装入内存运行【需要进行地址转换】

链接的3种方式：

1. 装入前链接：把所有模块和所需要的库函数链接成“绑定文件”

缺点：修改其中一个模块整体都要变造成难以修改；把所有可能不运行的模块都装入，浪费内存。

1. 装入时链接：B模块装入内存时检测可能需要的其它模块，连通它们一起链接并装入内存。（类比B：if C else D，则连同C,D都装入）

缺点：把B模块可能需要但是不一定执行的模块也装入内存链接，比如异常处理模块，但是B模块不发生异常就不需要它。

1. 执行时链接：A模块执行时发现一定要调用并执行B模块，这时B模块才装入内存并与A模块链接再执行（类比A：if B else C，则只装入B或C）

优点：模块独立，便于临时修改；节省内存；节省装入时间。

链接后的模块称为可执行模块，可执行模块装入主存后，模块中指令的地址是从0开始的相对地址，与物理地址不吻合，指令执行时，取数据的地址需要转换。

可重定位装入：分配的内存必须是连续的，链接好的模块装入内存后，立刻用实际的物理地址替换掉模块中的相对地址。程序执行期间，模块不能移动，也不能再申请主存空间。

动态运行装入：可以将程序分配到不连续的存储区，每当访问到相应指令或数据时，才把相对地址转换为物理地址。

装入时链接指系统装入含有未链接的外部模块引用的装入模块时，每当遇到一个外部模块引用，则查找相应的目标模块。将其装入内存，并将模块内的指令地址转换为相对于整个装入模块起始地址的相对地址。

优点：有利于目标模块的更新和升级，有利于代码共享，有利于扩充软件的功能，可以将扩充部分作为动态链接模块。

但是，可能链接一些不会执行的模块，浪费存储空间和处理机时间。

运行时动态链接指外部模块引用直至程序执行时才装入内存，并链接到装入模块中，进行地址转换。