内存管理

多层次存储结构

1. 可执行存储器：寄存器，高速缓存，主存，磁盘缓存

信息掉电即失，操作系统的存储管理机制对其分配，回收，提供数据移动管理机制

1. 固定磁盘，可移动存储介质

操作系统的设备管理机制管理

访问寄存器的速度和处理机处理的速度相匹配。

高速缓存用于保存主存中经常被访问的数据和暂存下一条待执行的指令。

处理机对可执行存储器和辅存的访问方式不同

磁盘缓存是主存中的部分用于暂存从磁盘中读出的信息的存储空间。

辅存中的数据必须复制到主存方可使用。

处理机从主存取得指令和数据，指令放入寄存器的指令寄存器，数据放入寄存器的数据寄存器。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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。

连续分配存储管理方式

单一连续分配：单道程序直接装入内存，相对地址就是绝对地址。

固定分区分配：把内存“切成”大小平均或不一的若干份，一份装一个程序

动态分区分配：连续的把内存切成n块大小不一的空闲分区，空闲分区用双向链表连起来，分配内存时，检测到某个适宜空闲分区，如果该空闲分区略大于申请内存直接把整个空闲分区分配给程序，如果大的很多，则切下多余的剩下的分给程序。

内存回收时，根据待归还的分区的首地址找到相应的插入点，回收区与插入点实际物理地址连续，则合并成一个空闲分区，否则，独立成一个空闲分区。

查找一块大小合适的空闲分区的算法：  
1，首次适应：每次都是从头到尾查询，碰到合适的就分配。容易造成低地址空间有很多容量很小的空闲分区，查找高地址的大空闲分区开销越来越大。

2，next 适应：新进程接着上次的分区往后查找大小适宜的空闲分区。避免了首次适应的查找开销，但容易逐渐造成空闲分区都很小，造成大程序无法装入。

3，最佳适应：空闲分区从小到大有序链接，这样找到的分区一定是切下来的多余空闲分区最少的分区。缺点：每次切下的分区都特别小，后面没法用。

4，最坏适应：把大于申请分区的最大空闲分区分配给进程，这样产生的“碎片”最少。【碎片】太小以至于绝大多数进程无法再利用的空闲分区。

为了提高搜索空闲分区的速度，大型系统采用基于索引搜索的动态分配算法。

快速适应算法，伙伴系统，哈希算法。

你能同步到电脑吗？