存储器管理

2021年11月4日 21:57

1、存储器的层次结构

1. 存储器的多层结构

   存储器至少有三级：CPU寄存器，主存，辅存

   层数越高，访问速度越快，价格越高，容量越小

   寄存器和主存储器又被称为可执行存储器

2. 主存储器与寄存器
3. 高速缓存与磁盘缓存

2、程序的装入和链接

程序运行需经过：编译、链接、装入

1. 程序的装入
   a. 绝对装入方式
      i. 程序编译后，产生绝对地址
      ii. 装入内存后，相对地址与实际内存地址完全相同
   b. 可重定位装入方式
      i. 根据内存的具体情况将模块装入到适当位置
      ii. 逻辑地址与实际物理地址不同
      iii. 地址变换在装入内存时一次完成，不再修改
      iv. 不允许程序运行时在内存中移动位置
   c. 动态运行时的装入方式
      i. 将地址变换推迟到程序真正要执行时才进行
      ii. 需要重定位寄存器
2. 程序的链接
   a. 静态链接方式

      程序运行前将各目标模块及他们所需的库函数链接成完整的装配模块，不再拆开

      1) 对相对地址进行修改
      2) 变换外部调用符号
   b. 装入时动态链接

      将编译后得到的目标模块边装入边链接

      1) 便于修改和更新
      2) 便于实现对目标模块的共享
   c. 运行时动态链接

      对某些模块的链接推迟到程序执行时

      只装入真正被使用的模块

3、连续分配存储方式

1. 单一连续分配

将内存分为系统区与用户区两部分，系统区提供给OS使用，用户区仅装有一道用户程序，整个用户区由该程序独占

2. 固定分区分配

   在系统中装入多道程序，这些程序之间不会发生相互干扰

   a. 划分分区的方式
      i. 分区大小相等
      ii. 分区大小不相等
   b. 内存分配

      根据内存大小检索分区使用表，找出一个能满足需求的尚未分配的分区

3. 动态分区分配
   a. 动态分区分配中的数据结构
      i. 空闲分区表
      ii. 空闲分区链
   b. 动态分区分配算法
   c. 分区分配操作
      i. 分配内存

         利用某种算法在空闲分区链（表）中找到所需大小的分区

      ii. 回收内存
         1) 与前一空闲分区F1邻接

            回收区与F1合并，修改F1大小

         2) 与后一空闲分区F2邻接

            F2与回收区合并，修改大小，使用回收区首地址作为新的首地址

         3) 回收区与F1，F2都邻接

            回收区、F2与F1合并，修改F1大小，撤销F2表项

         4) 既不与F1邻接，又不与F2邻接

            作为新分区插入到空闲分区链（表）

4. 基于顺序搜索的动态分区分配算法
   a. 首次适应FF

      从链首开始顺序查找满足大小的空闲分区，从分区中划分出空闲空间

      i. 倾向于使用低地址部分，高地址有大空闲区
      ii. 容易产生碎片
   b. 循环首次适应NF

      从上次查找的下一个分区开始查找

      i. 空闲分区分布均匀
      ii. 缺乏大的空闲分区
   c. 最佳适应BF

      使用满足需求的空间最小的空闲分区

      产生难以利用的碎片
   d. 最坏适应WF

      选择最大空闲分区，划分内存空间使用

      查找效率很高

5. 基于索引的动态分区分配算法

a. 快速适应QF
    i. 将空闲分区根据容量大小进行分类，每一类设置空闲分区链表
    ii. 根据进程长度，从对应链表中取下第一块进行分配
    iii. 分区归还主存时算法较为复杂，存在一定的浪费
b. 伙伴系统BS
    i. 所有分区的大小均为2^k
    ii. 对所有大小相同的分区设置双向链表
    iii. 对应大小的空闲分区耗尽时分割较大分区
    iv. 对于大小为2^k，地址为x的内存块，伙伴块的地址为

$$Buddy_k(x) = \begin{cases} x + 2^k & x|2^{k+1} = 0 \\ x - 2^k & x|2^{k+1} = 2^k \end{cases}$$

c. 哈希算法

6. 动态可重定位分区分配
    a. 紧凑

        将内存中的所有作业进行移动，使他们全部邻接，将原来的小分区拼接成大分区

        每次紧凑后需要对移动了的程序或数据进行重定位
    b. 动态重定位

        使用动态运行时装入方式时，当进行紧凑，无需对程序做任何修改，只要改变重定位寄存器的值，用新的始址置换原先的地址
    c. 动态重定位分区分配算法

        在动态重定位方法上加了紧凑功能

4、对换

1. 多道程序环境下的对换技术
    a. 对换的引入

        将内存中占时不能运行的进程调出到磁盘的对换区

        将磁盘上已具备运行条件的进程调入内存
    b. 对换的类型
        i. 整体对换

            以整个进程为单位
        ii. 页面/分段对换
2. 对换空间的管理
    a. 对换空间管理的主要目标
        i. 对文件区管理的主要目标

            提高文件存储空间的利用率，离散分配方式
        ii. 对对换区管理的主要目标

            提高换入和换出的效率，连续分配方式
    b. 对换区空闲盘块管理的数据结构
    c. 对换空间的分配与回收

        与内存分配方法相同
3. 进程的换入与换出
    a. 进程的换出

i. 选取被换出的进程
   1) 阻塞或休眠的进程
   2) 内存驻留时间
   3) 优先级
ii. 进程换出过程

b. 进程的换入
   i. 就绪态进程
   ii. 换出到硬盘的时间

5、分页存储管理方式

离散空间分配，无需执行紧凑

分页存储管理

分段存储管理

段页式存储管理

1. 分页存储管理的基本方法
   a. 页面和物理块

      进程的逻辑地址分成多个页，内存的物理地址分成多个块

      分配内存时，以块为单位，将进程的若干个页装入到多个可以不相邻的物理块中

   b. 地址结构

      页号+位移量（页内地址）

   c. 页表

      实现页号到物理块号的映射

2. 地址变换机构
   a. 基本的地址变换机构
      i. 页表寄存器
      ii. 分页地址变换机构
   b. 具有快表的地址变换机构
      i. 页表存放在内存中，CPU读取数据时需要两次访问内存
      ii. 为提高地址变换速度，在地址变换机构中增加具有并行查找能力的高速缓冲寄存器快表
      iii. 快表项可以直接从快表中取出物理块号
3. 访问内存的有效时间

      t：访存时间 λ：查找快表时间 a：快表命中率

   a. 不设有快表：$EAT = t + t = 2t$
   b. 设有快表：$EAT = a \times \lambda + (1 - a) \times (\lambda + t) + t = 1(+ (1 - a)t) + \lambda$
4. 二级页表和多级页表
   a. 二级页表

      寻址空间很大时，难以找到大的连续内存空间存放页表

      引入外层页表，将页表离散的存储在内存中

      外层页号+外层页内地址+页内地址

      三次访存

b. 多级页表

5. 反置页表
   a. 减少页表占用的内存空间
   b. 为每一个物理块设立页表项，内容为页号及所属进程标识符
   c. 根据进程标识符与页号检索反置页表

6、分段存储管理方式

1. 分段存储管理方式的引入
   a. 方便编程
   b. 信息共享
   c. 信息保护
   d. 动态增长
   e. 动态链接
2. 分段系统的基本原理
   a. 分段

      段号+段内地址
   b. 段表

      每个分段分配一个连续的分区

      段基址+段长
   c. 地址变换机构

      越界错误

      段表数目一般比页表数目少
   d. 分页和分段的主要区别
      i. 页是物理单位，目的在于提高内存利用率。段是逻辑单位，目的在于保存完整信息
      ii. 页的大小固定，段的大小不固定
      iii. 分页的用户程序地址是一维的，分段的用户地址是二维的（段长信息）
3. 信息共享
   a. 分页系统

      不同进程页表项指向相同页面
   b. 分段系统

      不同进程段表项指向相同段

      实现更为便捷
4. 段页式存储管理方式

   将分段划分为分页

   段号+段内页号+页内地址

   需要同时配置段表和页表