第 7 章 主存管理

月出皓兮　苏曙光老师的课堂笔记

2022 年 4 月 24 日

目录

1 三级存储体系

理想中的存储体系：容量足够大，速度足够快，信息永久保存，多道程序并行（共享与保护问题）。

实际中我们使用三级存储体系，当内存太小不够用时，用辅存来支援内存。**换入-换出策略（地址重定位问题）**。

Cache(快, 小, 贵) + 内存（适中, 信息易丢失）+辅存（慢, 大, 廉, 可保存）

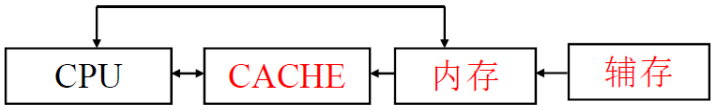

图 1: 三级存储体系

2 存储管理的功能

2.1 地址映射

2.1.1 定义

把程序中的地址（虚拟地址, 虚地址, 逻辑地址）变换成真实的内存地址（实地址, 物理地址）的过程。

虚拟地址/源程序包括地址，变量，标号，函数名

2.1.2 固定地址映射

编程或编译时确定逻辑地址和物理地址映射关系。程序加载时必须放在指定的内存区域。容易产生地址冲突，运行失败。不能适应多道环境。

例如：中断向量处理表的位置。

2.1.3 静态地址映射

程序装入时由操作系统完成逻辑地址到物理地址映射。

保证在运行之前所有地址都绑定到主存。程序装入后不能移动，如果移动必须放回原来位置。

逻辑地址：VA(Virtual Addr.)
装入基址：BA(Base Addr.) 基址寄存器 BAR
物理地址：MA(Memory Addr.)

$$MA = BA + VA$$

2.1.4 动态地址映射

在程序执行过程中把逻辑地址转换为物理地址。

程序占用的内存空间可动态变化。程序不要求占用连续的内存空间。共享代码作为独立的一段存放。其需要硬件支持，而且编写的软件较为复杂。

逻辑地址：VA(Virtual Addr.)
装入基址：BA(Base Addr.) 基址寄存器（重定位寄存器）BAR
物理地址：MA(Memory Addr.)

$$MA = BA + VA$$

硬件实现动态地址映射：重定位寄存器作为进程运行现场的一部分。进程换入进入运行态时，重定位寄存器要恢复。

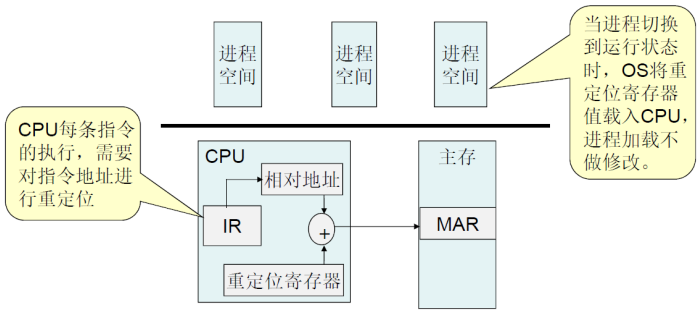

图 2: 硬件实现动态地址映射

连续进程空间在不连续的物理块存放，多个基址寄存器（重定位寄存器）——段式存储分配技术。

按段编译，虚拟地址格式：段 + 段内位移，不同段放入不同的物理块。

2.2 虚拟存储（内存扩充）

基本问题是：程序过大或过多时，内存不够，不能运行。
解决方案是借助辅存在逻辑上扩充内存，解决内存不足。
迁入：把在辅存存放的部分临时按需调入内存。
迁出：把当前不运行的部分暂时存放在辅存上
用户体验了“足够大的线性内存”—虚拟内存

2.2.1 依据：程序局部性原理：时间局部性，空间局部性

程序在一个有限的时间段内访问的代码和数据往往集中在有限的地址范围内。因此，一般情况下，把**程序的一部分装入**内存在较大概率上也足够让其运行一小段时间。

2.2.2 前提条件

足够的辅存
适当容量的内存
动态地址映射机制

2.2.3 应用

页式虚拟存储和段式虚拟存储。

2.3 内存分配

2.3.1 放置策略

程序调入内存时将其放置在哪个/哪些内存区?
按需分配/存储扩充

2.3.2 调入策略

何时把要运行的代码和要访问的数据调入内存
需要访问时，调度进入内存

2.3.3 淘汰策略

迁出（/淘汰）哪些代码或数据以腾出内存空间。

2.4 存储保护（考试不做要求）

保证在内存中的多道程序只能在给定的存储区域内活动并互不干扰。共享问题和保护问题对立统一。

防止访问越界，防止访问越权。

2.4.1 界址寄存器法

在 CPU 中设置一对下限寄存器和上限寄存器存放程序在内存中的下限地址和上限地址。判断是否越界。

基址寄存器和限长寄存器

适于连续物理分区中的情形

2.4.2 存储键保护法

适于不连续物理分块的情形，也可用于共享中的权限。

3 分区存储管理系统

3.1 单一区存储管理

用户区不分区，完全被一个程序占用。例如：DOS。

简单，不需复杂硬件支持，适于单用户单任务 OS。但程序运行占用整个内存，内存浪费，利用率低。

3.2 固定分区存储管理

定义：把内存固定地划分为若干个大小不等的分区供各个程序使用。每个分区的大小和位置都固定，系统运行期间不再重新划分。

例如：IBM 的 OS/360

浪费内存，且大程序可能无法运行。

3.2.1 分区表

记录分区的位置、大小和使用标志

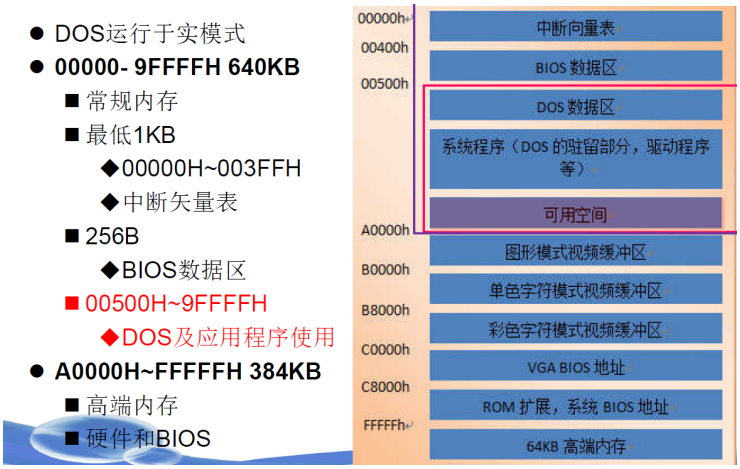

图 3: DOS 操作系统内存管理

3.3 动态分区存储管理

定义：在程序装入时创建分区，使分区的大小刚好与程序的大小相等。

分区的个数和大小均可变，但存在**内存碎片（过小的空闲区，难实际利用）**。

3.3.1 空闲区表与空闲区队列

描述内存空闲区的位置和大小的数据结构

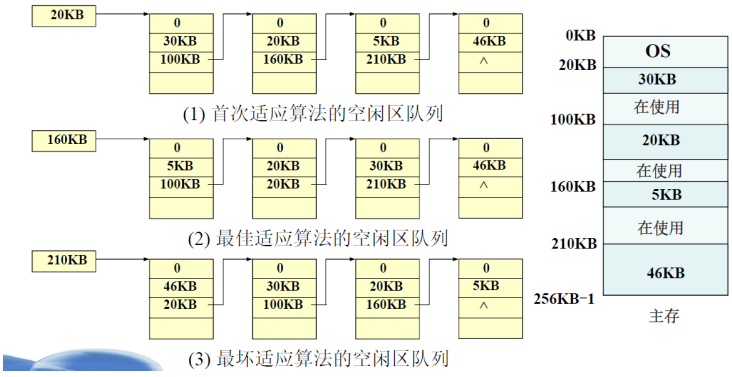

图 4: 空闲区表与空闲区队列

3.3.2 放置策略：首次适应算法

空闲区表按首址递增排序，尽可能先利用低地址空间。

3.3.3 放置策略：最佳适应算法

空闲区表按大小递增排序，尽量选中满足要求的最小空闲区。

3.3.4 放置策略：最坏适应算法

空闲区表按大小递减排序，尽量使用最大的空闲区。
这样产生的内存碎片较少。

3.3.5 分配策略：一般将底部分配给用户

从用户选中的分区中分配/分割所需大小给用户, 剩余部分依然作为空闲区登记在空闲区表中。

分割空闲区时一般把**底部**分割给用户。以减少改动的参数量。

3.3.6 回收策略

回收程序占用的分区（释放区), 将其**适当处理**后登记到空闲区表中, 以便再分配。

插入，更新，删除。

3.3.7 碎片问题解决方法：规定门限值

分割空闲区时，若剩余部分小于门限值，则此空闲区不进行分割，而是全部分配给用户。

3.3.8 碎片问题解决方法：内存拼接技术

将所有空闲区集中一起构成一个大的空闲区。

当释放区回收的时候，或系统找不到足够大的空闲区时进行拼接。

空闲区的管理复杂，消耗系统资源；离线拼接；

其有一种改进策略：重新定义作业。把程序分拆几个部分装入不同分区，充分利用碎片。解除程序占用连续内存的限制。

3.4 覆盖技术——overlay

目的：在较小的内存空间中运行较大的程序

内存分区：

常驻区：被某段单独且固定地占用的区域，可划分多个

覆盖区：能被多段共用（覆盖）的区域，可划分多个

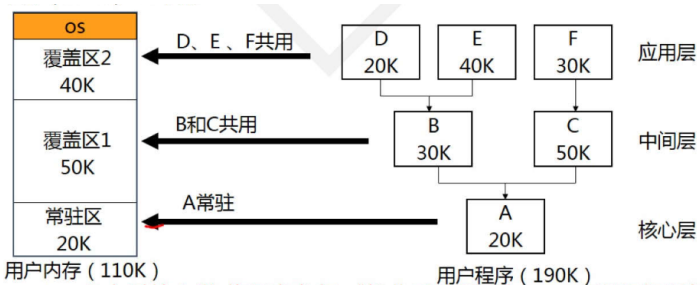

图 5: 覆盖技术应用举例

缺点：编程复杂，程序员需根据分块，划分程序模块并确定覆盖关系。

3.5 对换技术——swapping

原理：内存不够时把进程写到磁盘（换出/Swap Out ）。当进程要运行时重新写回内存（换入/Swap In ）。

优点：增加进程并发数；不考虑程序结构。

缺点：换入和换出增加 CPU 开销；交换单位太大（整个进程)

需要考虑的问题：减少交换传送的信息量（模块/段)；外存交换空间的管理方法；程序换入时的地址重定位

4 虚拟内存管理

4.1 概念

虚拟内存是**面向用户的虚拟封闭存储**空间（进程空间)。

1. 线性地址空间
2. 容量 4G $= 2^{32}$ Byte（32 位系统下）
3. 封闭空间（进程空间）

4. 和物理地址分离（地址无冲突）
5. 程序员编程时使用线性虚拟地址（映射到物理内存）

4.2 实现思路

1. 代码分段
2. 逐段装入
3. 逐段运行，逐段迁出
4. 和物理地址分离（地址无冲突）
5. 时间换空间

4.3 经典虚拟内存管理方式

1. 页式虚拟存储管理
2. 段式虚拟存储管理
3. 段页式虚拟存储管理
4. 和物理地址分离（地址无冲突）
5. 时间换空间

5 页式存储管理系统

5.1 概念

概念：把进程空间（虚拟）和内存空间都划分成等大小的小片
进程的小片——页（虚拟页或页面或虚页）
内存的小片——页框（物理页或块或实页）
小片的典型大小：1K，2K 或 4K…

5.2 地址映射机制

5.2.1 页面映射表（页表）

记录页与页框 (也称块) 之间的对应关系。也叫页表。

操作系统为每个进程建立一个页表，页表长度和首址存放在**进程控制块**中。

当前运行进程的页表驻留在内存，页表长度和首址由**页表长度寄存器和页表首址寄存器**指示。

主要记录：

1. 页号：登记程序地址的页号。

2. 页框号：登记页所在的物理页号。

3. 中断位 I 和辅存地址：中断位 I 标识该页是否在内存，若 I =1，不在内存。

4. 访问位（引用位）和修改位（Dirty）.

5.2.2 虚拟地址

页大小和页框大小一般相等，设为 N。

虚拟地址 VA 线性，从 0 开始，可以分为页号 P 和页内偏移 W。

$$P = VA/N$$

$$W = VA\%N$$

查页表后得到 P 对应的页框号 $P^{'}$。

$$MA = P^{'} \times N + W$$

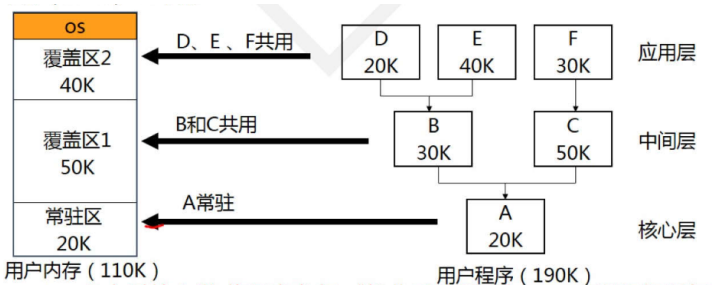

图 6: 页式地址映射实现

5.2.3 二级页表机制

将 4M 的页表以页为单位分成若干个小页表，存入离散的若干个页框中。

为了对小页表进行管理和查找，另设置一个叫页目录的表，记录每个小页表的存放位置（即页框号）。

小页表含 1K 条记录。小页表大小 4K。小页表占一个页框。小页表总共有 1K 个。大页表大小 4K。

设页表数为 M，页框大小与页大小为 N。页表号 S，页表框号即为 $S^{'}$。页号为 P，页框号即为 $P^{'}$。

$$S = VA/(M \times N)$$

$$P = (VA/N)\%M$$

$$W = VA\%N$$

$$MA = P^{'} \times N + W$$

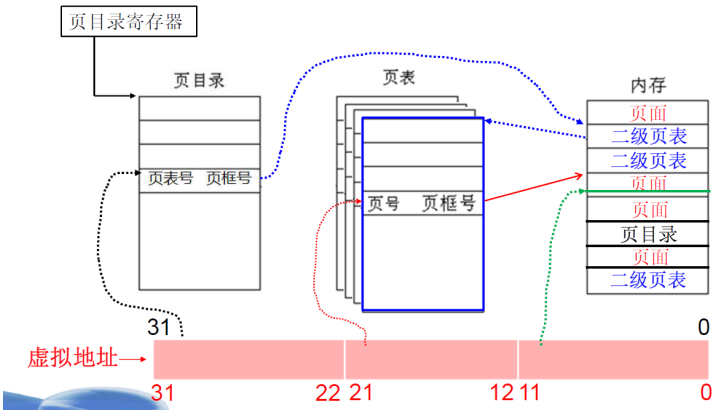

图 7: 二级页表的地址映射过程

5.2.4 快表机制（CACHE，TLB）

快表是普通页表（慢表）的部分内容的复制。地址映射时优先访问快表. 若在快表中找到所需数据，则称为“命中”。没有命中时，需要访问慢表，同时更新快表.

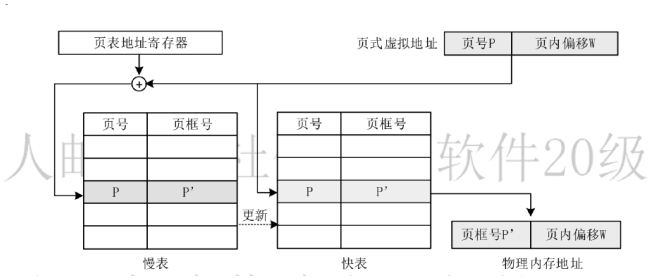

图 8: 快表机制的实现

5.2.5 命中率

合理的页面调度策略能使快表具有较高命中率

5.3 页面共享机制

例：.dll 动态链接库
使用同一个页框。

5.4 内存分配机制

5.4.1 缺页中断

在地址映射过程中，当所要访问的目的页不在内存时，则系统产生异常中断——缺页中断。

其与普通中断的不同为，普通中断在指令完成后响应，而缺页中断在指令执行过程中发生。一条指令执行时可能产生多个缺页中断

局部性越好，越不容易缺页跳转或分支越多越容易缺页

缺页（中断）率：缺页率 f = 缺页次数/ 访问页面总次数

5.4.2 淘汰策略（OPT 算法，FIFO 算法，LRU 算法，LFU 算法）

最佳算法（OPT 算法）

淘汰不再需要或最远将来才会用到的页面。

理论上最佳，实践中该算法无法实现。

先进先出淘汰算法（FIFO 算法）

淘汰在内存中停留时间最长的页面。

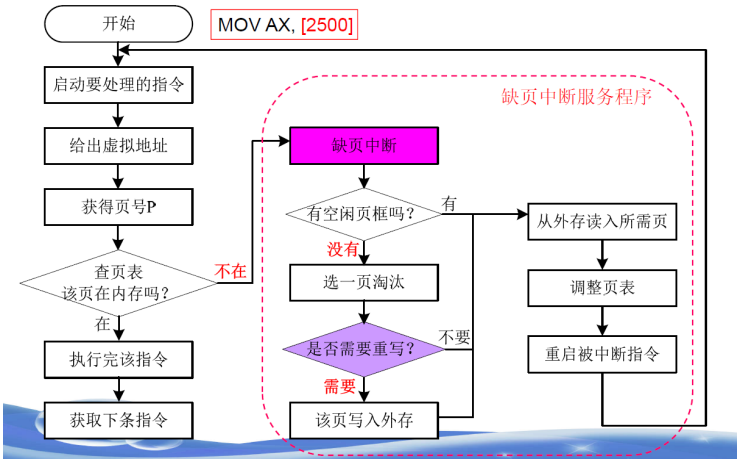

图 9: 缺页中断的产生与处理

其存在异常现象。对于一些特定的访问序列，分配页框越多，缺页率越高!

最久未使用淘汰算法（LRU 算法）

淘汰最长时间未被使用的页面。

图 10: 硬件方法实现 LRU 算法

最不经常使用（LFU 算法）

选择到当前时间为止被访问次数最少的页面

每页设置访问计数器，每当页面被访问时，该页面的访问计数器加 1；

发生缺页中断时，淘汰计数值最小的页面，并将所有计数清零。

6 段式存储管理系统

6.1 概念

以段为单位装入，每段分配连续的内存；但是段和段不要求相邻。

段需要连续的存储空间; 段的最大尺寸受到内存大小的限制；在辅存中管理可变尺寸的段比较困难；

6.2 地址映射机制

6.2.1 段表（SMT，Segment Memory Table）

记录每段在内存中映射的位置
段表的字段:
段号 S： 段的编号（唯一的）
段长 L： 该段的长度
基地址 B： 该段在内存中的首地址
扩展字段：中断位，访问位，修改位，R/W/X

6.2.2 虚拟地址

段式虚拟地址 VA 包含段号 S 和段内偏移 W。
检索段号 S，查询该段基地址 B 和长度 L。

$$MA = B + W$$

6.3 段页式存储管理

在段式存储管理中结合页式存储管理技术。在段中划分页面。

6.3.1 地址映射机制：同时采用段表和页表

系统为每个进程建立一个段表；
系统为每个段建立一个页表；
段表给出每段的页表基地址及页表长度（段长）。

页表给出每页对应的页框。

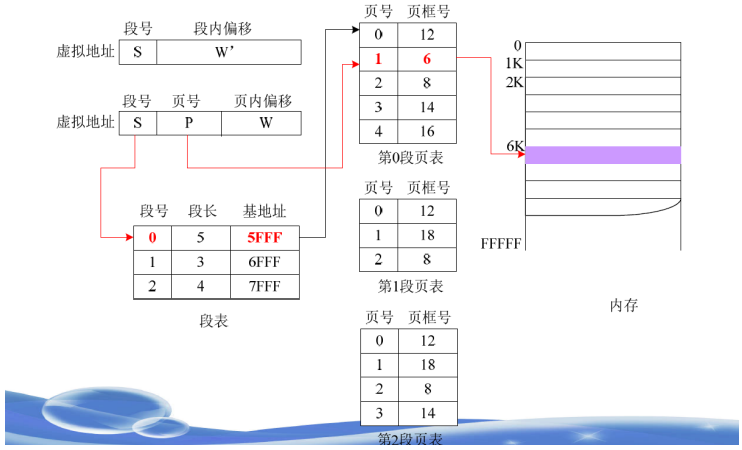

图 11: 段页式存储管理的地址映射机制

7 Linux 的/proc 文件系统

内存文件系统。

为用户访问内核信息提供接口，如：CPU 信息，内核信息（如进程信息），物理地址信息。

7.1 /proc/cpuinfo

处理器信息，如类型、制造商、型号和性能。

7.2 /proc/数字

关于某个进程的信息目录。

cat 以手工查看

程序中可以文件 IO 访问

在 shell 程序中用 cat 命令结合正则表达式来获取并处理内核信息.

cmdline 该文件包含的是该进程的命令行参数，包括进程的启动路径(argv[0])。