存储管理

概论

存储管理

存储管理是操作系统的重要组成部分，负责管理主存储器，主要包括

- 主存储空间的分配和去配
- **地址转换和存储保护**
- 主存储空间共享
- 主存储空间扩充

存储管理主要是管理主存中的**用户区域**

地址转换

源程序生成可执行程序需要经过

- 程序编译
- 程序链接
  - 静态链接：装载入内存前就已将目标模块链接入程序
  - 动态链接：一边装载一边链接
  - 运行时链接：将链接推迟到执行到目标代码时
- 程序装载
  - 绝对装载，静态地址重定位：指令地址与内存中的地址相同，地址为内存绝对地址，由装载程序实现将程序装入指定位置，无需硬件支持也不能移动程序位置
  - 可重定位装载，动态地址重定位：根据内存情况决定装入位置，模块内部是相对地址，起始地址装入重定位寄存器，由**硬件实现**逻辑地址到物理地址的转换，称为动态地址重定位
  - 动态运行时装载，运行时地址重定位：允许将装入内存的程序换出到磁盘，适当时候再换入内存，即允许进程的内存映像在不同时刻不同，模块内部为相对地址，运行时动态地址重定位，动态将其链接

存储保护

目标：防止操作系统和用户程序在主存中各区域访问时互相干扰

存储保护硬件：有界地址和存储键

连续存储空间管理

程序占据主存中的连续空间

单用户连续存储管理

任何时刻主存中只有一道程序。将主存划分为系统区和用户区

系统利用率低

固定分区存储管理

称为定长分区或静态分区模式，满足多道程序，基本思想为给进入主存的用户作业划分一块连续存储区域，把作业装入该连续存储区域，若有多个作业装入主存，则它们可并发执行

系统启动时静态地将可分配的主存分割为若干连续区域，每个区域位置固定，任何时刻一个区域只能装入一道程序

存储保护方面，若是静态定位，则检查其绝对地址是否在分配的分区内，动态定位时，使用地址上下限寄存器检查是否越界

调度作业时选择可容纳其的最小可用分区

缺点在于难以预知分区大小，且分区数目预先确定，限定了多道程序的数量

可变分区存储管理

称为变长分区模式，基本思想是按作业大小划分分区，划分的时间、大小、位置均**动态确定**，装入程序前系统不预先分区

将主存分为使用部分和未使用部分，分别建立链表，可以使用起始地址和分区长度标识一个分区。当装入作业时寻找一个可容纳其的空闲分区，将其分为两部分：使用和空闲（如果有），将使用部分的长度和起始地址加入已分配表，修改未分配表中的起始地址和长度

程序退出内存时需要考虑可变分区的回收问题，即将分区合并。需考虑左右分别为已分配和未分配的情况

常用分配算法

最先适应（first fit）：顺序查找未分配表，直到找到第一个满足要求的空闲区，通常未分配链表表项按地址从小到大排列，会给分区回收带来复杂度，且高低地址利用不均衡

下次适应（next fit）：从未分配表上次结束的位置继续搜索，使存储空间利用更平衡

最优适应（best fit）：扫描未分配表，寻找满足条件的最小的分区。空间利用率高，若将未分配表按照分区长度组织，等于最先适应算法。查找时间长，难以利用分配后的小碎片

最坏适应（worst fit）：寻找最大的未分配区分配，使剩下的未分配区不至于过小，对中小型作业有利

快速适应（quick fit）：为常用的长度的空闲区单独建立表，如 2KB 空闲区，4KB 空闲区等，归还空间时复杂度很高

地址转换与存储保护

一般对每个分区设置两个寄存器，基址寄存器和限长寄存器，存储分区的起始地址和长度，以此来进行地址转换和存储保护

提供多对基址/限长寄存器的系统中允许一个进程占据多个分区，可以实现内存共享

伙伴内存分配（Buddy memory allocation）

基本思想是将用户区域对半分割，实现最佳适应。采用这种分配方式外碎片很小，且对于分区的合并也很方便，但是分区内部存在内碎片

应对内存不足的连续存储管理技术

连续存储管理的系统经过一段时间的运行，会产生许多碎片（不连续的小容量分区）

移动技术

将内存中的进程分区连接到一起，将分散的未分配区汇聚。开销很大，现代操作系统不使用

覆盖技术

基本思想为将大程序划分为一系列的覆盖，每个覆盖是一个相对独立的程序单位，把程序执行时不需要同时装入内存的覆盖构成一组，称为覆盖段。一个覆盖段内的覆盖共享同一存储区域，该区域成为覆盖区，其大小由对应的覆盖段内最大的覆盖决定。

交换技术

将暂时不用的程序的一部分或全部从内存移到外部存储器

分页式存储管理

基本原理

分页式存储的基本原理是允许作业存放在若干个不相邻的分区中，充分利用主存空间，尽量减少主存碎片。

分页式存储中的基本概念有

- 页框：主存空间按物理地址划分为多个大小相等的分区，每个分区称为页框（page frame）
- 页面：程序按逻辑地址分为多个大小相等的区，每个区称为页面（page），大小与页框相等
- 逻辑地址：逻辑地址被划分为两部分，高位指示页号，低位指示页内偏移量
- 内存页框表：表中为内存中的页框，给出其使用情况，以及保护位等
- 页表：将逻辑页面与物理页框一一对应的查找结构

分页存储中的地址转换过程为：将逻辑地址划分为页面号和页内偏移量，根据页面号在页表中查询物理页框号，则物理地址为页框号*页框大小+页内偏移量

系统中有一个页表基址寄存器，当前占用 CPU 的进程在其中存放自己的页表基址，当进程切换时作为上下文保存

快表与多级页表

快表

页表项如果放在硬件寄存器中，实现时硬件代价过高，而如果放在内存中时，每次访问地址操作都会带来两次访存（访问页表+访问实际物理地址），因此在寄存器中存放最近被访问的部分页表，称为快表。

当访存时先查询快表中是否有对应页面，如果没有则再访存查询页表

多级页表

页表规模过大，考虑页面大小为 4KB 时的情况，则在 32 位系统中共有 2^{20} 个页面，设每个页表项占存储空间为 4B，则一个进程的页表大小就达到了 4MB，这显然是不可接受的。一种解决方案是使用多级页表

二级页表系统中，将逻辑地址划分为三部分，从高位到低位为页目录号，页表号，页内偏移量，地址转换时先根据页目录号在页目录中查询对应页目录的基址，然后在页目录中查询对应页的页框号，最后根据页框号*页框大小+页内偏移量获得物理地址

令每个二级页表大小为一个页面的大小，程序运行时页表按需调入内存即可，内存中只用维护一个页目录，减小了内存消耗，如 32 位系统页面大小为 4KB，则一个页表令其大小为一个页面大小，每个表项大小为 4B，则可容纳 2^{10} 个页，地址中高 10 位为页目录索引，中 10 位为页表索引，低 12 位为页内偏移量

反置页表

反置页表的出现也是为了解决页表占用空间过大的问题。一般页表根据逻辑页号查询物理页框号，而反置页表根据物理页框号查询页号，即仅保存内存中的页面信息，大小与主存大小相关（页表大小与虚存大小相关）。

反置页表中表项存储物理页框对应的逻辑页号和进程标识符，实际使用时使用一个哈希函数，以进程标识和页号作为输入，输出反置页表的表项，根据此即可获得物理地址。当多个输入映射到同一输出时，使用链表链接起所有的结果。当未找到结果时即触发缺页异常

存储空间分配去配

分页式存储的存储空间分配以页框为单位，需要建立内存物理块表记录页框的分配状态

最简单的方法是使用位示图记录分配情况，每位对应一个页框，分配为 1，未分配为 0

也可以使用链表记录未分配的页面，分为分配链表和空闲链表，每个表节点记录分配/未分配区块的起始块号和页框数量，内存回收时只用修改两个邻居区块的表节点即可

页面共享与页面保护

多个进程共享程序时，要求每个逻辑地址的页号是唯一确定的，这样可以使不同进程里的逻辑页映射到同一物理页框

共享信息的保护可以在页表中增加标志位，指示该页的访问权限，或是使用存储保护键法

分段式存储管理

分段式存储管理是为了满足程序设计和开发的需求（以模块为单位的装配、共享和保护）

基本原理

分段式存储管理的逻辑地址划分为段号和段内位移，该地址结构是用户可见的。分段式存储管理基于可变分区存储管理，为每个段分配一段连续的空间。为程序建立段表，段表包含各段的段号，段起始地址和段长度，实际上起到了基址/限长寄存器的作用。

地址转换时根据段号查询段表，获得段起始地址，根据段长进行存储保护

段的共享通过不同程序的段表中的项指向同一段基址实现

与分页的对比

段是信息的逻辑单位，是由源程序的逻辑结构决定的（代码段，数据段，栈段等），分段是对用户可见的，段长由用户决定，段基址可从任何地方开始。源程序的地址为二维结构

页是信息的物理单位，与源程序无关，对用户也是透明的，页长由系统决定，且起始位置必须对齐。源程序的地址为一维的结构

虚拟存储管理

基本原理

虚拟存储器的基本思想就是为了提高主存利用率，程序运行时只把当前运行所需的部分程序/代码装入内存，其余部分存放在辅助存储器，需要时调入

- 部分装入：当访问到不在主存的数据时，需要系统将信息装入主存
- 部分对换：当主存中没有空间时把暂时不会用到的信息从主存调出，移动到辅存

这样的话，多个程序需要的主存空间大于主存时也可运行，且使得主存的物理大小对用户透明，允许逻辑地址空间大于物理空间

虚拟存储器：在具有层次结构存储器的计算机系统中，采用自动实现**部分装入**和**部分对换**功能，为用户提供一个比物理主存容量大得多的，可寻址的一种“主存储器”

虚拟存储器能实现的理论基础是程序的局部性，包括空间和时间

不同于对换技术，虚拟存储管理以页或段为单位处理，且在进程所需存储大于主存时仍能运行

虚拟存储管理需要解决的问题有

- 主存与辅存统一管理
- 逻辑地址到物理地址的转换
- 部分装入与部分对换

请求分页式存储管理

原理

请求分页式存储管理是分页式存储管理的扩展，基本原理是将程序分为多个页面，存放在辅存中，当程序运行时，仅装入访问使用的页面，运行过程中若需要访问的页面不在主存则将其装入（请求分页中的"请求"）

请求分页的页表项扩充了标志位，一般来说至少包含

- 引用位：页面被引用（读/写）时设置，用于页面淘汰
- 修改位：页面被修改后置位，被修改的页面调出时必须写回磁盘
- 保护位：限定页面的访问权限
- 驻留标志位：指示页面是否装入内存，未装入内存时访问该页面产生缺页异常，根据磁盘地址将该页面调入内存，页面与磁盘地址对应的表称为外页表

硬件支持

请求分页式管理依靠底层硬件的支持，即主存管理单元 (MMU, Memory Management Unit)

MMU 的功能是完成逻辑地址到物理地址的转换，以及在此过程中产生硬件中断（缺页/越界），MMU 主要由页表基址寄存器和快表组成，功能为

- 管理页表基址寄存器
- 分解逻辑地址为页号和页内偏移量
- 管理快表，包括查询和装入/清除表项，具体而言，每次 TLB 不命中，查询页表后将相应表项填入 TLB，在页表基址寄存器修改时（进程切换）也要清空 TLB
- 访问页表，根据页表查询结果将逻辑地址转换为物理地址
- 发出异常，若访问页表时缺页或是访问越界则发出硬件中断
- 管理特征位，即管理页表中表项的各个特征位

装入与调出策略

页面装入有两种基本策略：请页式和预调式

- 请页式：缺页中断驱动，每次装入一页
- 预调式：按某种预测算法动态预测并装入若干页

页面调出也有两种基本策略，请页式和预约式

- 请页式：仅当一页被选中替换时，该页内容若修改则写回辅存
- 预约式：内容被修改的页面成批写回辅存，写回在替换前而不是替换时

还有一种页缓冲技术，被淘汰的页面进入两个队列，修改和非修改。修改队列的页定时被成批写回并加入非修改队列，而非修改队列的页面，或是因为引用而离开队列，或是被淘汰用作页面替换

页面分配策略

页框分配的原则为尽量减少缺页中断

为每个进程分配一定数量的页框，若分配较少，主存中可容纳的进程数就会增加，但是进程的缺页率也会上升，而若分配给进程的页框超过一定限度后，因为程序本身的原因，不会明显再降低缺页率。

页面分配可分为固定分配与可变分配

- 固定分配：在进程的生命周期中分得的页框数固定
- 可变分配：缺页率较高则分配较多页框，反之分配较少页框

而当缺页后要调入新页面，页面替换也可分为局部替换与全局替换

- 局部替换：页面替换算法范围仅限于进程分得的页框，一般和固定分配结合使用
- 全局替换：页面替换算法范围为整个系统的页框，一般和可变分配结合使用

一般设进程 P 总访问页面次数为 A，其中成功访问次数为 S，缺页次数为 F，则缺页率为

$$f = \frac{F}{A}$$

抖动现象是指处理器花费大量时间用于处理缺页异常而不是正常计算

一般一个页面替换算法要求尽量避免抖动现象并且有尽可能低的缺页率

缺页率受以下因素影响

- 页框数：进程分得的页框越多，缺页率越低
- 页面大小：页面大则缺页率较低
- 页面替换算法
- 程序特性：主要指程序的空间/时间局部性

典型页面替换算法

最佳页面替换算法（OPT, Optimal replacement）

即每次淘汰时淘汰以后不再访问的页，或是距现在最长时间后才访问的页面。

OPT 算法没有实用性，因为难以预测一个程序的页面访问记录，但是给定一个页面访问记录，OPT 算法可以用来衡量一个具体可实现的页面替换算法的优劣

随机页面替换算法

随机决定要替换的页面，实现简单但性能差

先进先出页面替换算法（FIFO）

总是淘汰最先调入主存的页面（即驻留时间最长的页面）

FIFO 算法实现简单，但是仅适合具有线性特性的程序，对其余程序效率不高

> Belady 异常：增加分配页框数有时候不会增加 FIFO 的性能

第二次机会页面替换算法（SCR, Second Chance Replacement）

改造 FIFO 算法，与引用位结合使用，具体实现时检查队列的队首

- 若其引用位为 0，其最早进入内存且最长时间未使用，选择其淘汰
- 若其引用位为 1，说明其虽然最早进入内存但最近仍有使用，将其引用位清零，并且将其移至队列尾

SCR 实现较为简单，且效率高

时钟页面替换（Clock, Clock policy replacement）

使用标准队列时，SCR 会产生频繁的入队出队，代价较大。可使用循环队列构建页面的队列

- 页面装入内存时引用位置 1
- 页面被访问时引用位置 1
- 淘汰页面时，从指针当前指向循环队列的位置开始扫描
  - 若指向页面引用位为 1，则置为 0，指针前进
  - 若指向页面引用位为 0，则淘汰该页面

考虑页面被修改的情况，修改后淘汰页面需写回辅存，开销大于淘汰未修改的页面，故将引用位和修改位结合即可得到改进的 Clock 算法

页面共有 4 种可能情况

- $r=1, m=1$，最近被引用，且被修改
- $r=1, m=0$，最近被引用但未被修改
- $r=0, m=1$，最近未被引用，但是被修改
- $r=0, m=0$，最近未被引用，且未被修改

于是替换过程可描述为

1. 扫描队列，途中**不改变**引用位，淘汰遇到的首个 $r=0, m=0$ 的页面
2. 若扫描完后没有淘汰，再次从原位置开始，查找 $r=0, m=1$ 的页面，若有则将其淘汰，同时将扫描到页面的引用位置为 0
3. 若仍未淘汰，从 1 开始，此时由于所有页面的引用位都为 0，必能淘汰符合 1 或 2 条件的页面

最近最少用页面替换（LRU, Least Recently Used replacement）

淘汰最近一段时间内最久未被访问的页面，需要维护一个页面淘汰队列，使得队尾指向最近使用的页面，队头即为要淘汰的页面

这样实现缺页率低，但是代价也很高（每次访问页面都要调整队列），故一般采用模拟 LRU 的方法

- 标志位法，又称最近未使用页面替换法（NRU, Not Recently Used replacement），每次访问页面时将其引用位置 1，间隔时间 t 周期性将所有引用位清零。淘汰页面时从引用位为 0，即最近 t 时间内未引用的页面中选择页面替换
- 计数法，又称最不常用页面替换算法（LFU, Least Frequently Used replacement），每次页面被引用时将其引用计数器+1，经过时间间隔 t 后将所有计数值清零，淘汰页面时选择引用次数最少的页面替换
- 计时法，每次引用页面时记录绝对时间，经过时间间隔 t 后将其清零。淘汰页面时选择计时值最小的页面淘汰
- 老化算法，为每个页设置一个多位寄存器，当页面访问时将最左位置为 1，经过时间间隔 t 后所有寄存器右移一位。当淘汰页面时，选择寄存器数值最小的页面，即最不常用的页面替换。此算法思想类似指数加权移动平均，最近发生的事件影响最大

上述模拟算法的难点都在于信息更新的周期 t 难以设定

局部最佳页面替换算法（MIN, local Minimum replacement)

与 OPT 算法类似，需要预先知道页面访问顺序。若进程在 t 时刻访问页面缺页，则将其加入空闲页框，若此页面在 $(t, t+\tau)$ 的时间内未被再次引用，则将其调出，否则保留，直至再次被引用。τ 是个常量，$(t, t+\tau)$ 被称为滑动窗口

增加 τ 可以降低缺页率，但代价是花费更多页框

工作集置换算法（WS, Working Set replacement)

概念同滑动窗口，但并非向前而是向后（基于局部性原理）

进程工作集指在某一阶段内进程运行所需访问页面集合，用 $W(t, \Delta)$ 标识进程在 $(t-\Delta, t)$ 间访问的页面集合，称为进程在时刻 t 的工作集，Δ 为工作集窗口大小

只有属于工作集的页面才能留在内存，替换思想类似 MIN

工作集替换算法开销很大，一般采用引用位结合时间戳实现模拟算法，即周期性检查引用位

- 若引用位为 1，将其置为 0，记录下时间戳
- 若引用位为 0，记录下系统时间到记录时间戳的差 t_{off}，若 $t_{\text{off}} > t_{\max}$ ，淘汰该页

其中 $t_{\max}$ 为系统中设定好的常量

或是使用老化算法模拟

缺页频率替换算法（PFF, Page Fault Frequency replacement)

即根据连续的缺页之间的时间间隔来对缺页频率进行测量，每次缺页时，利用测量时间调整进程工作集尺寸。

如果本次缺页与前次缺页之间的时间超过临界值 T，那么，所有在这个时间间隔内没有引用的页面都被移出工作集。

相比工作集算法实现起来代价低，仅在缺页时修改工作集

页面大小

请求分页式管理的页面大小设定时，主要需考虑

- 页表大小
- 主存利用率
- 读写页面的时间（对于磁盘为移臂+旋转，大页面平均读写时间短）和传输时间

最佳页面的尺寸在 512 B 到 8 KB 之间，过小会导致页表过大，且读写时间增加，过大会导致页面内碎片增加，最佳页面尺寸估计为

$$f(p) = \frac{se}{p} + \frac{p}{2}$$

其中 p 为页面大小，s 为进程平均大小，e 为页表项大小

请求段页式虚拟存储管理

结合段式存储管理和分页存储管理的优点

- 段式存储管理，有利于模块化程序设计，但段之间的碎片浪费存储空间
- 页式存储管理，有利于提高存储空间的利用率

基本思想为

- 虚地址以逻辑结构划分为段
- 实地址分为大小位置固定的页框
- 线性地址空间划分为和页框大小相等的页面

即逻辑地址经过段表转换为线性地址，再经过页表转换为实际的物理地址