虚拟存储器

1. 常规存储器管理方式局限性：

1 一次性：作业在运行前必须一次性地全部装入内存后方能开始运行，

2 驻留性：作业装入内存后，会一直驻留在内存中，直至作业运行结束。

----一些暂时不用的程序和数据占据大量内存空间，使得一些需要运行的作业无法装入运行。

----一个程序所要求的内存空间超过了内存实际容量，则无法装入内存

2. 局部性原理：

1 时间局部性：程序中存在大量的循环操作，使得刚刚访问过的指令和数据不久后还要再次访问。

2 空间局部性：程序在一段时间内所访问的地址，可能集中在一定的范围内，其典型情况便是程序的顺序执行。

3. 虚拟存储器----基于局部性原理

1 定义：具有请求调入功能和置换功能、能从逻辑上对内存容量加以扩充的存储器系统称为虚拟存储器。

2 特征：

1) **多次性**，一个作业分成多次调入内存。

多次性是虚拟存储器最重要的特征

2) **对换性**，允许将那些暂不使用的程序或数据从内存调至对换区，待以后需要时再调入内存

有效地提高内存利用率

3) **虚拟性**，虚拟存储器对内存的扩充是逻辑上的，用户所看到的大容量只是一种感觉，并不实际存在，因此是虚拟的。

虚拟性是实现虚拟存储器的目标

3 注意：

1) **虚拟存储器必须建立在离散分配的基础上**

2) 实现方式也可分成请求分页、请求分段和请求段页式

4 硬件支持：需要硬件支持，必须提供请求分页（段）的页（段）表机制，以及缺页（段）中断机构和地址变换机构

5 软件支持：需要软件支持，请求调页（段）的软件以及实现页（段）置换的软件

请求分页存储管理方式

1. 支持虚拟存储器的请求分页系统：分页+请求调页功能+页面置换功能。
   1. 每次调入和换出的基本单位都是固定长度的页
   2. 目前最常用的实现虚拟存储器的方式
2. 请求分页的基本原理。

请求分页系统要求将作业的部分页面装入内存，开始运行作业，其余部分被存放在磁盘中。请求分页系统的硬件提供了请求页表机制，加了以下内容：

（1）状态位 P，用于指示该页是否已调入内存，以供程序访问时参考。

（2）访问字段 A，记录本页在最近一段时间内被访问的次数或最近已有多长时间未被访问，置换算法在选择换出页面时参考。

（3.）修改位 M，调入内存后是否被修改过，供换出页面时参考，决定是否需要将换出页重新写回外存。

（4）外存地址，指出该页在外存上的地址，供调入页面时参考。

3. 基本原理：

在请求分页系统中，当进程需要访问某条指令或某个数据时，硬件地址变换机构将根据逻辑地址中的页号去检索内存中的页表，并根据相应页表项的存在位 P 来判断该指令或数据所在的页是否已装入内存，若已装入内存，则可立即从页表项中得到该页的内存块号。并与页内地址拼接形成指令或数据的物理地址，同时还需修改页表项中的访问位，对于写指令，则还需将修改位置成“1”。若所要访问的页还未调入内存，便产生一缺页中断，此时，上述访问缺页的作业将被中断，控制将转向缺页中断处理程序。

缺页中断处理程序用来完成页面的调入工作。若系统中仍有空闲的内存块，则只需根据页表项中的外存地址将所缺的页装入内存，然后修改页表项中的存在位和内存块号即可；否则，若系统中无空闲的内存块，则需要根据置换算法淘汰内存中的某一页，对已被修改过的淘汰页则还需要先将其写入磁盘，然后再将所缺的页调入内存

4. 内存分配策略和置换策略

固定分配和可变分配。全局置换和局部置换。

(1) 固定分配局部置换策略：

   1. 进程分配的物理块数目，在进程的整个生命周期都固定不变；
   2. 因调入页面二需要划出某个页面，能换出它自己的内存页面。

基本原理.

① ~~请求~~

① 页面请求 → 逻辑地址 → 页号 → 判断该页是否加入内存

查询页表（快表）

→ 页内地址

(B) 修改状态位、访问位：1

A> 加入内存 → 获取物理块号 → 物理地址 → 访存

B> 未加入内存 → 缺页中断 → 调用缺页中断处理程序

② 缺页中断：调入页面.

A> 内存有空闲：将页面从外存 → 内存

B> 内存不够：置换：

i> 调出内存中某个页

ii> 将缺页调入内存

页表修改状态位，确定是否写回外存

页面请求 → 逻辑地址：页号 + 页内地址 W

→ 页号≥页表长度 → Y：越界中断

N：

→ 1> 查快表找状态位 P：

判断该页是否在内存

Y：页号 → 内存块号(b)：物理地址：b×L+W

→ 访存（附注：状态位 所改字段）

N：

→ 缺页中断

→ 缺页处理程序：

→ 1> 内存中有空闲块：将缺页调入内存

1> 内存中没有 ——：页面置换（淘汰 调入）

淘汰页面是否写回外存

调入缺页

因内存块太少而频繁缺页，因内存块太多而浪费空间。

(2) 可变分配全局置换策略：

① 每个进程分配一定数目的物理块

② 当进程发生缺页时，若系统中有空闲的物理块，则为其分配一个物理块并装入缺页

③ 若系统中已没有空闲的物理块没从内存中选择一页换出，再装入缺页，换出页无意是系统中任一进程的页，又会使那个进程的物理块减少，进而使其缺页率增加。

(3) 可变分配局部置换策略。

① 每个进程分配一定数目的物理块后，若某个进程发生缺页，将自己的某个内存页换出。

② 进程在运行中频繁发生缺页中断，则系统须为该进程分配若干附加物理块，直至其缺页率减少到适当程度为止。

优点：可变分配局部置换策略可获得较高的内存空间利用率，又能保证每个进程有较低的缺页率。

5. 调页策略

(1) 请求调页策略

在运行中需要访问某部分程序和数据是，若发现其所在的页面不在内存，立刻发出缺页中断，求 OS 将所需页面调入内存。

(2) 预调页策略

预调页策略指，将那些预计自不久之后要回访问到的几个页面，预先调入内存的策略。

由于预测哪些页面在不久之后便会访问是十分困难的，其成功率只有百分之 50%，故预调页策略主要用于进程首次调入和整体换入时。

置换算法

在无空闲的内存块时，若要调入某个缺页，便必须将内存中的某个页面换出到磁盘对换区，用来选择换出页面的算法被称作置换算法。下面将介绍几种常用的置换算法。

1. 最佳(OPT)置换算法

保证获得最低的缺页率，由于无法预知哪个页是未来最长时间内不被访问的，该算法只能是一种理论上的算法。

用来评价其他算法的优劣。

2. **先进先出（FIFO）置换算法**

FIFO 算法总是选择最先进入内存的页面予以淘汰。

实现简单，往往与进程实际运行规律不相符。

3. 最近最久未使用（LRU）算法及其近似算法。

LRU 算法赋予每个页面一个访问字段，用来记录相应页面自上次被访问以来所经历的时间 t，当淘汰一个页面时，应选择所有页面中其 t 值最大的页面，即内存中最近一段时间内最长时间未被使用的页面。LRU 算法利用“最近的过去”作为“最近的将来”的近似，由于程序访问的时间局部性，它一般能有较好的性能，但为了快速地判断哪一页是最近最久未用的页面，它需要较多的硬件支持，会增加系统的成本，故在实际应用中，大多只采用 LRU 的近似算法。

Clock 算法就是一种常用的 LRU 近似算法，它为每个页设置一位访问位，再将内存中的所有页面通过链接指针链成一个循环队列。当某页被访问时，其访问位由硬件置 1。置换算法从替换指针开始顺序检查循环队列中的各个页，如果其访问位为 0，就选择该页换出并将替换指针指向下一个页面；若访问位为 1，则将它置为 0，并继续向下查找。由于该算法只有一位访问位，只能用来表示一页最近是否被访问过，并选择最近未被访问过的页面作为淘汰页，故又称为最近未用（NRU）算法。

如果一个页在换入内存后曾被修改过，则换出时需要将它写入磁盘，否则就不必写入磁盘。考虑到这种置换代价，可将上面的 Clock 算法作以下改进：首先从替换指针开始第一次扫描循环队列，并选择首个访问位和修改位都为 0 的页面进行换出；如果扫描一轮仍没找到换出页，则开始第二轮扫描，并选择首个访问位为 0、修改位为 1 的页面进行换出，扫描过程中将所有扫描过的页面的访问位置 0；如果仍未找到换出页，则重新开始上面的第一轮扫描，必要时再重复第二轮扫描，便一定能找到被换出的页

4. 最少使用（LFU）置换算法

LFU 算法选择最近一段时间内，内存中访问次数最少的页面进行淘汰。LFU 算法的实现同样需要得到较多硬件的支持，而且对一个新调入的页面，可能会因它的访问次数最少而被淘汰，而另一些页面因为在某个时候被访问多次，即使以后不再使用，也不会马上被淘汰，从而使 LFU 算法性能不佳，因此该算法并不常用。

5. 页面缓冲算法 PBA

采用 PBA 算法时，被换出的页面仍留在内存的空闲块中，所有的空闲块形成一个空闲页面缓冲池。因此，发生缺页时，如果能从空闲页面缓冲池中找到所缺的页，则直接可将对应的物理块分配给进程而无需启动磁盘 I / O，否则，需要为缺页分配一个空闲块并将所缺的页读入其中；同时还需按照某种算法（如实现非常简单的 FIFO 算法）选择一个淘汰页，该淘汰页所占用的内存块被作为空闲块，加入到空闲页面缓冲池中。有的系统，如 VAX / VMS 系统，还根据淘汰页是否需要写回磁盘，而把空闲页面缓冲池中的空闲块组织成空闲页面链表和修改页面链表两个链表，每次分配内存块时，总是将空闲页面链表中的第一个物理块分配出去，而当修改页面链表中的空闲块数达到一定数量时，再将它们一起写回磁盘，以减少磁盘 I / O 的次数，提高系统效率。

置换算法的好坏将直接影响到系统的性能，不适当的算法可能会导致进程发生“抖动”，即刚被换出的页很快又要被访问，为此，又要换出其他页，而该

页又很快被访问，如此频繁地置换页面，以致大部分的时间都花在页面的置换上。通常，可通过调节内存中多道程序的度来控制“抖动”的发生。

请求分段存储管理方式

1. 请求分段的基本原理

与请求分页系统类似，在分段的基础上增加请求调段功能和段置换功能，便可形成具有虚拟存储器功能的请求分段系统。

请求分段系统的段表中也需增加状态位 P、访问字段 A、修改位 M、外存地址等内容，它们的含义与请求分页系统中相同，在允许分段动态增长的系统中，段表中还设有增补位，用来表示每个段在运行过程中，是否做过动态增长。

请求分段系统的地址变换机构，是在分段系统地址变换机构的基础上形成的。若段表项中的存在位指示段在内存，则可根据段的内存基址和段内地址形成物理地址，并修改段表项中的访问位和修改位；如果分段不在内存，则将发出缺段中断，请求 OS 将所缺的段调入内存，并在相应段调入内存后，再利用已修改的段表进行地址变换。在调入所缺的段时，若内存中有足够大的空闲分区，则可根据段表中的外存地址直接装入分段；否则，若空闲分区总和能满足需要，则可在紧凑后将所缺的分段装入内存；如果空闲分区的总和也难以满足需要，则必须根据置换算法淘汰一个或几个段，以形成一个足够的空闲分区，再将所缺的分段装入内存。

2. 分段的共享

为了实现分段共享，可在系统中配置一张共享段表，每个共享段都在共享段表中占一表项。共享段表项中记录了共享段的具体信息，如：段名、段长、内存始址、状态（存在）位、外存始址以及共享该段的进程引用计数等；还记录有共享该段的所有进程的信息，如：进程名、进程号、共享段在进程中的段号、进程对该段的存取控制权限等。

首次被请求时，系统会为共享段分配一物理区，再把共享段调入该区，同时将该区的始址，填入请求进程的段表的相应项中，还须在共享段表中增加一表项，填写共享段的信息和请求使用该共享段的进程的信息，其引用计数为 1。以后，当又有其他进程需要调用该共享段时，则无须再为该段分配内存，而只需在调用进程的段表中，填写该共享段的物理地址，并在共享段表的对应表项中填上调用进程的信息，引用计数增加 1。当共享段被释放时，需对引用计数进行减 1 操作，只有当引用计数为 0 时，才能回收该共享段的物理内存，并取消其在共享段表中的表项。基于索引书点的共享方式

3．分段的保护

在分段系统中，由于每个分段在逻辑上是相对独立的，因而比较容易实现信息保护。目前，常采用以下几种措施，来确保信息的安全。

（1）越界检查。在分段系统中，地址变换机构将比较逻辑地址中的段号与段表寄存器中的段表长度，以及逻辑地址中的段内地址和段表项中的段长，如果段号太大或段内地址太大，都将发生越界中断。

（2）存取控制检查。分段系统的段表项中设有存取控制字段，用来规定相应段的访问权限（如只读、只执行、可读／写等），在地址转换过程中，将自

动检查本次操作是否与存取控制字段中的访问方式相符，若不相符，则发出保护性中断信号。

（3）环保护机制。采用环保护机制的计算机系统中，CPU 可以有多种执行状态，每种状态具有不同的特权，因此形成多个特权环。通常较低编号的环具有较高的特权，并规定一个程序可以访问驻留在相同环或特权更低的环中的数据，可以调用驻留在相同环或特权更高的环中的服务。这样，只要将程序和数据安排在不同的特权环内，如 OS 核心处于 0 环内、某些重要的系统软件占据中间环、普通用户程序安排在外环上，便可对信息进行有效的保护。

① 在一采用局部置换策略的请求分页系统中，分配给某个作业的内存块数为4，其中存放4个页面的情况如下：

M=4

302!

物理块	虚页号	装入时间	最后一次访问时间	访问位	修改位
0	2	60 ✓	157 4	0	1
1	1	160 ✓	161 2	1	0
2	0	26 ✓	158 3	0	0
3	3	20 ✓	163 1	1	1

所有时间都是从进程开始运行时，从0开始计数的时钟数

采用下列置换算法，将选择哪一页进行换出？

① FIFO算法 —— 物理块3的第3页

② LRU算法 —— 物理块0，虚页号2 ✓

③ 改进的Clock算法 —— 物理块2，虚页号0 —— 访问位0，修改位0

下次换为

② 在一个请求分页系统中，如果一个作业的页面走向：

4，3，2，1，4，3，5，4，3，2，1，5　　　M=3　M=4

目前没有任何页装入内存，当分配给该作业的物理块 M 分别是 3 和 4 时，分别计算采用 LRU 和 FIFO 页面淘汰算法时的缺页次数和缺页率。

比较两种置换策略在不同物理块下的性能？

① LRU　M=3　　缺页　置换

页面：	4	3	2	1	4	3	5	4	3	2	1	5
缺页：	√	√	√	√	√	√	√			√	√	√
置换：				4	3	2	1			5	4	3

~~4 3 2 1 4 3 5 4 3~~ 2 1 5

缺页次数：10

缺页率 = 10/12 = 5/6

② LRU M=4

页面	4	3	2	1	4	3	5	4	3	2	1	5
命中	√	√	√	√			√			√	√	√
缺页												
置换												

3 2 1

4 3 2 1 5

缺页次数：8

缺页率 = 8/12 = 2/3

③ FIFO　　M=3

页面：	4	3	2	1	4	3	5	4	3	2	1	5
缺页：	√	√	√	√	√	√	√			√	√	
置换：				4	3	2	1			4	3	

4	3	2	1	4	3	5	2	1

缺页次数：9

缺页率：9/12 = 3/4

④ FIFO $M=4$

页面：	4	3	2	1	4	3	5	4	3	2	1	5
	√	√	√	√			√	√	√	√	√	√
缺页：							4	3	2	1	5	4
置换：												

~~4 3 2 1 4 3~~ 5 4 3 2 1 5

缺页次数：10

缺页率 $= 10/12 = \frac{5}{6}$

③ 某页式虚拟存储系统中，页面大小 1KB，某进程分配到内存块数为3，M=3
按下列地址引用内存单元：

3635　3632　1140　3584　2892　3640　0040　2148　1700
2145　3209　0000　1102　1100　　逻辑地址

上述地址均为十进制，且内存中尚未装入任何页面。

给出采用 LRU 算法时的缺页次数，并与 FIFO 算法进行比较。

1) 计算出页面号 → 页面访问顺序

3635/1024 = 3　3632/1024 = 3　1140/1024 = 1　3584/1024 = 3
2892/1024 = 2　3640/1024 = 3　0040/1024 = 0　2148/1024 = 2
1700/1024 = 1　2145/1024 = 2　3209/1024 = 3　0000/1024 = 0
1102/1024 = 1　1100/1024 = 1

页面访问顺序：3 3 1 3 2 3 0 2 1 2 3 0 1 1

LRU M=3

页面访问: 3 3 1 3 2 3 0 2 1 2 3 0 1

缺页: √ √ √ √ √ √ √ √

置换:

3	3	3	3	3	3	3	2	2	2	2	2	1
		1	1	1	0	0	0	1	1	3	3	3
				2	2	2	2	2	3	3	0	0

缺页=8

FIFO M=3

页面: 3 3 1 3 2 3 0 2 1 2 3 0 1

缺页: √ √ √ √ √ √ √ √

3	3	3	3	3	3	0	2	1	2	3	0	1
		1	1	1	1	3	3	2	2	2	3	3
				2	2	2	0	0	0	1	1	0

缺页=6

改进:

[例⑤] 某虚拟存储器用户空间共32个页面，每页1KB，主存16KB，[假定某时刻]系统为用户的第0、1、2、3页分配物理块号5、10、4、7，

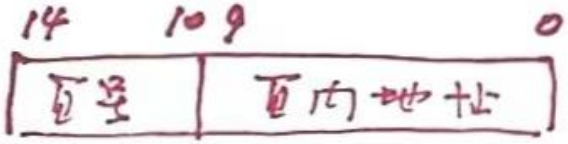
设用户作业长度为6页，试将十六进制虚拟地址：

0A5C　　103C　　1A5C 转换成物理地址。

① 逻辑地址：32个页面 → 2^5 个

页面大小：1KB → 2^{10} B

⇒ 用15位二进制表示一个地址



② 主存16KB，每页1KB ⇒ 16KB ⇒ 2^{14}

$2^{14} \Rightarrow 2^4 \cdot 2^{10}$

物理地址：14位

前4位：块号

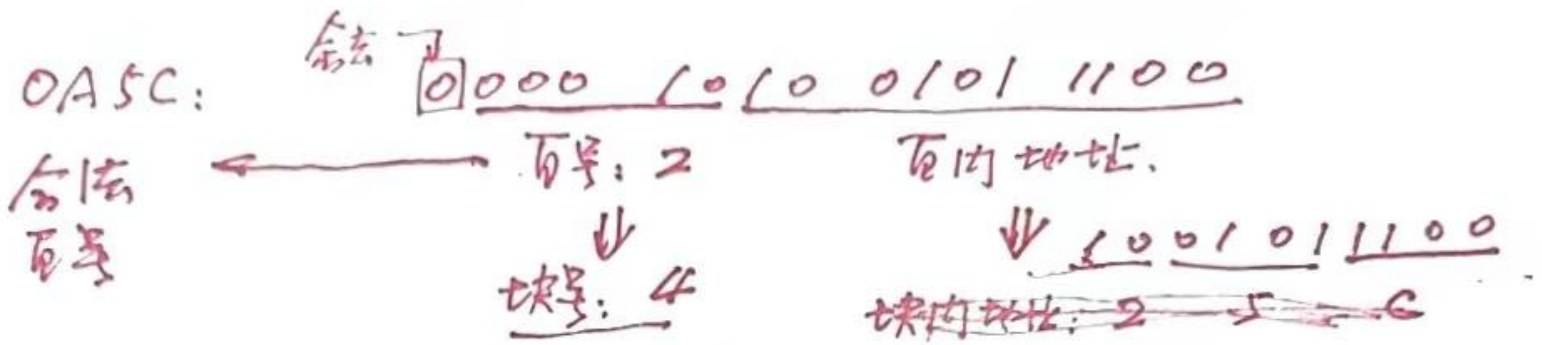
后10位：块内地址

③ ~~计算~~：

~~0A5C~~

页号	内存块
0	5
1	10
2	4
3	7

① 0A5C：



~~块内地址：2 5 C~~

物理地址：块号：0100

~~拼加~~：~~块内地址：~~

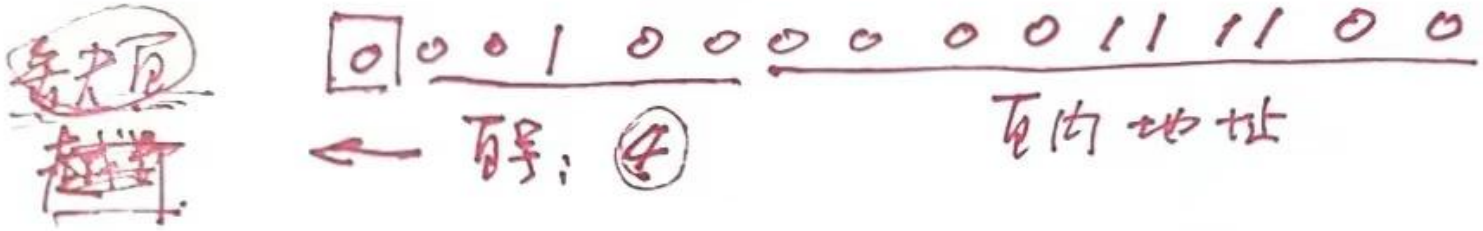
二进制物理地址：0100 1001011100

十六进制物理地址：1 2 5 C

物理地址：125C

~~缺页中断~~

② 103C → 十六进制：



103C → 越界（页表越界）. 0. 1. 2. 3

↳ 4号页在外存

↳ 缺页中断

1A5C 二进制:

[0] 001 10 10 0101 1100

越界异常 ← 页号: ⑦ (0.1.2.3 页表, 4.5 外存)

1A5C — 越界异常