块地址转换开销的对换区，其 I/O 效率会比文件区更高一些。但对换区资源有限，需要系统按照不同的情况，采取合适的策略来合理使用这一资源。

(a) 系统拥有足够多的对换区，则在进程运行前就将其所有文件拷贝到对换区，后续缺页中断时全都从对换区调入，以提高 I/O 效率[3]。

(b) 系统缺少足够多的对换区，则优先将更可能被修改的文件放入对换区。因为文件被修改过后，调出就需要写磁盘，连续写可以有效降低这一代价。而那些不可能被修改的文件，由于不需要写回磁盘，则直接从文件区调入[3]。

(c) UNIX 方式。未运行过的页面都存放在文件区并直接从文件区调入，换出的页面才放入对换区，后续再次用到时则从对换区调入[3]。

3.2.4 页面置换方法

置换页面的完整过程可以概括为以下三步：

- ◇ 根据当前为该进程分配的页框数和其已使用页框数，判断是否页框已满。根据相关策略，确定是否置换。当所有页框都已被使用：
  - ✧ 若为可变分配，则视情况选择额外分配页框或者置换。
  - ✧ 若为固定分配，则需要置换。
- ◇ 确定置换范围：
  - ✧ 局部置换，只能从属于当前进程的页面中选取。
  - ✧ 全局置换，既可以从当前进程页面中选取，也可从其他用户进程页面中选取。
- ◇ 按照一定的页面置换方法，选取一个合适的页面。

其中，页面置换方法是置换功能的核心，其性能直接决定了虚拟存储器的运行效率。下面来详细介绍这些算法。

1. 最佳置换算法（Optimal page replacement algorithm, OPT）

最佳置换算法是指在每一次置换中都把将来不会再使用或者最长时间不再使用的页面调出。这是最高效的算法，但很明显的问题是，系统并不能预知接下来需要用到哪些页面，所以无法实现该算法。虽然这一算法不能实现，但其可以作为衡量调入算法性能的标杆。

此方法相关习题在习题精编中有收录，见节末习题精编。

2. 先进先出（First In First Out, FIFO）页面置换算法

FIFO 算法是指每次置换都将最早被调入的页面调出内存。该算法是基于队列实现的，存在 Belady 异常。即在一定的情况下，增加为进程分配的页框数，缺页率反而会增加。如下例所示。

【举例】访问队列为：{1、2、3、4、1、2、5、1、2、3、4、5}，按照分配页框数 3 和 4，分别计算缺页中断次数。

(1) 分配 3 个页框时，如表3.8所示，最终结果是共发生了 9 次缺页中断。

(2) 分配 4 个页框时，如表3.9所示，最终结果是共发生了 10 次缺页中断，这比只分配 3 个页框时还多出 1 次缺页中断，即发生了 Belady 异常。

并且 FIFO 没有利用上局部性原理，其性能较差，所以几乎不会被使用。

[3]汤小丹 & 汤子瀛. 计算机操作系统（第 4 版）. 西安电子科技大学出版社. 161 页.

表 3.8 分配 3 个页框时缺页中断计算过程

No.	处理页面	内存中(按调入时间排序)	当前缺页次数	No.	处理页面	内存中(按调入时间排序)	当前缺页次数
1	1	1	1	*7*	5	1、2、5	7
2	2	1、2	2	*8*	1	1、2、5	7
3	3	1、2、3	3	*9*	2	1、2、5	7
4	4	2、3、4	4	*10*	3	2、5、3	8
5	1	3、4、1	5	*11*	4	5、3、4	9
6	2	4、1、2	6	*12*	5	5、3、4	9

表 3.9 分配 4 个页框时缺页中断计算过程

No.	处理页面	内存中(按调入时间排序)	当前缺页次数	No.	处理页面	内存中(按调入时间排序)	当前缺页次数
1	1	1	1	*7*	5	2、3、4、5	5
2	2	1、2	2	*8*	1	3、4、5、1	6
3	3	1、2、3	3	*9*	2	4、5、1、2	7
4	4	1、2、3、4	4	*10*	3	5、1、2、3	8
5	1	1、2、3、4	4	*11*	4	1、2、3、4	9
6	2	1、2、3、4	4	*12*	5	2、3、4、5	10

3. 最近最久未使用（Least Recently Used, LRU）置换算法

LRU 算法是指将最长时间未被使用的页面置换出内存。

最佳置换算法性能优秀，但无法实现，究其原因就在于无法预知未来有哪些页面将会被 CPU 访问。但可以将局部性原理运用于算法之中来预测未来一段时间较有可能被使用的页面有哪些。基于时间局部性，最近有被使用过的页面很有可能还会被使用，自然在做页面置换时应当将那些最久未被使用的页面换出内存。这就是我们所说的 LRU 置换算法。

下面以一个例题来对 LRU 算法的执行过程做详细说明。

【例 3.2】系统为某进程分配了 4 个页框，该进程已访问的页号序列为 2，0，2，9，3，4，2，8，2，4，8，4，5。若进程访问的下一页的页号为 7，依据 LRU 算法，应淘汰页的页号是（ ）。

A. 2　　B. 3　　C. 4　　D. 8

解：选 A。对于考查 LRU 算法的题目可以使用如下表格来进行解题。表格的最后一列描述的是当前页面被处理过后，内存中所存在的页面，并且这是一个按照最后访问时间排列好的队列。处理当前页面从而生成最后一列内容的方式总结如下：

① 若当前页面已存在于内存中，则将当前页面调整至队尾，其他部分不变。

② 若当前页面不在内存中，则将当前页面加入到队尾，若加入当前页面后内存中页面数大于页框数，则将队头页面移除，该页面则为被置换出内存的页面。

实际解题时主要需要记录的只有该表的最后一列，前三列是为了保证题解的清晰程度额外书写的。

No.	已处理页面	当前处理页面	内存中 (排列原则：最后访问时间)
1	2	2	2
2	2、0	0	2、0
3	2、0、2	2	0、2
4	2、0、2、9	9	0、2、9
5	2、0、2、9、3	3	0、2、9、3
6	2、0、2、9、3、4	4	2、9、3、4
7	2、0、2、9、3、4、2	2	9、3、4、2
8	2、0、2、9、3、4、2、8	8	3、4、2、8
9	2、0、2、9、3、4、2、8、2	2	3、4、8、2
10	2、0、2、9、3、4、2、8、2、4	4	3、8、2、4
11	2、0、2、9、3、4、2、8、2、4、8	8	3、2、4、8
12	2、0、2、9、3、4、2、8、0、4、8、4	4	3、2、8、4
13	2、0、2、9、3、4、2、8、2、4、8、4、5	5	2、8、4、5
14	2、0、2、9、3、4、2、8、2、4、8、4、5、7	7	8、4、5、7

LRU 算法利用局部性原理，通过页面使用历史来对未来的页面使用做预测，需要使用页表项中的访问字段。虽然该算法没有最佳适应算法效果好，但其可实现，并且相比 FIFO 置换算法，LRU 的性能有了非常明显的提升。但是 LRU 算法实现困难，系统开销大。

【拓展】LRU 算法可以用软件方式实现，也可以依靠硬件实现，但不论采用哪种方式其带来的系统开销都比较大。以下对这两种实现方式各举一简单例子来加以说明：

软件方式实现：系统维护一个双向链表，表头是最近使用页面，表尾是最近最久未使用页面。这个链表在每次访问页面之后都需要进行更新。置换时，选择表尾页面换出。

硬件方式实现：系统设置一个计数器，每个页面需要留出空间用来写入计数器的值。系统每执行一条指令计数器就加 1，并且将计数器中的数字写入页面的预留位置。置换时，遍历所有页面，找出记录计数器值最小的页面换出。

4. 最少使用（Least Frequently Used, LFU）置换算法

LFU 算法是指将最近使用次数最少的页面置换出内存。

LFU 算法与 LRU 算法的思想相同，其区别只在于，LRU 关注的是某一页面上一次使用的时间，而 LFU 关注的是页面在最近一段时间内的使用频率。

5. 简单的时钟（Clock）置换算法

时钟置换算法将所有页面链接成一个循环队列，每个页面置一个访问位，每次该页面被访问到，都将访问位设为 1。当发生置换时，查询指针前进一位。若所指向的页面访问位为 0，则换出该页。若访问位为 1，则将其置为 0，且查询指针前进一位。循环往复直到找到第一个访问位为 0 的页面换出，实际上一次换出最多进行两轮扫描。

下面以一个具体的例题来帮助理解。

【例 3.3】若某进程最多需要 6 页数据存储空间，操作系统采用固定分配局部置换策略为此进程分配 4 个页框。在时刻 260 前的该进程访问情况如表3.10所示（访问位即使用位）。

页号	页框号	装入时刻	访问位
0	7	130	1
1	4	230	1
2	2	200	1
3	9	160	1

表 3.10　访问表

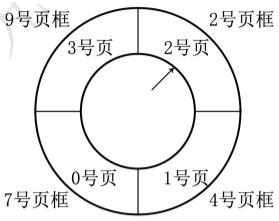

图 3.30　CLOCK 示意图

当该进程执行到时刻 260 时，要访问 5 号页内的数据。请回答：

若采用时钟（CLOCK）置换算法，此次访问是否会发生缺页中断？其 5 号页面对应的页框号是多少？要求给出计算过程（设搜索下一页的指针沿顺时针方向移动，且当前指向 2 号页框，如图3.30所示）。

解：页面 5 未被调入内存，所以会引发缺页中断以调入该页面。此时进程被分配到的页框只有 4 个，已被全部占满，需要进行页面置换。

根据 CLOCK 算法寻找换出页面，从 2 号页开始顺时针扫描。

(1) 指针指向 2 号页面，2 号页面访问位为 1，将访问位置 0 继续扫描。

(2) 指针指向 1 号页面，1 号页面访问位为 1，将访问位置 0 继续扫描。

(3) 指针指向 0 号页面，0 号页面访问位为 1，将访问位置 0 继续扫描。

(4) 指针指向 3 号页面，3 号页面访问位为 1，将访问位置 0 继续扫描。

(5) 指针指向 2 号页面，2 号页面访问位为 0，选择 2 号页面换出，扫描结束。

所以 5 号页面被装入到 2 号页面对应的 2 号页框中。

LRU 算法的效果好，但是需要较多的硬件支持，实现起来也比较复杂。CLOCK 算法则是 LRU 的一种近似算法，它并不追求严格的最近最久未使用，可以认为是将最近较久未访问的页面换出。自然，CLOCK 算法实现起来相对较简单，不需要太多的硬件支持。

6. 改进的时钟（Clock）算法

未被修改的页面被调出时不需要进行写磁盘操作，置换代价更小，可考虑优先被置换。为此可以在页表项中增加一个修改位，用来表示该页面在被调入内存之后是否有被修改过。将内存中的页面分为如下 4 类，并按照如下的优先级来选择调出页面（P 表示访问位，M 表示修改位）：

(1) 第一类 (P = 0, M = 0) 页面：最近未被访问且未被修改，优先级最高。

(2) 第二类 (P = 0, M = 1) 页面：最近未被访问但被修改，优先级次之。

(3) 第三类 (P = 1, M = 0) 页面：最近有被访问但未被修改，换出优先级第三。

(4) 第四类 (P = 1, M = 1) 页面：最近有被访问且被修改，优先级最低。

在具体的实现中，系统要寻找的是第一类 (P = 0, M = 0) 或第二类 (P = 0, M = 1) 页面，然后将其换出。最多将所有页面循环扫描四轮即可成功找到要换出的页面。具体流程如下：

第一轮：寻找第一类 (P = 0, M = 0) 页面，找到第一个即换出，未找到则进行第二轮。

第二轮：寻找第二类 (P = 0, M = 1) 页面，找到第一个即换出，并且每遇到一个非第二类 (P = 0, M = 1) 页面都将该页面的访问位置为 0，若未找到则进行第三轮。

第三轮：继续寻找第一类页面，找到第一个即换出，未找到则进行第四轮。

【提示】由于第二轮查找时修改了访问位，这一轮实际上所找的是最初的第三类页面。

第四轮：继续寻找第二类页面，找到第一个即换出，这一轮的查找是一定能成功的。

【提示】由于第二轮查找时修改了访问位，这一轮所找的实际上是最初的第四类页面。

表 3.11　页面置换算法总结

方法	概述	优缺点
OPT	置换时把将来不会再使用或者最长时间不再使用的页面调出。	性能最好，但是无法实现，可作为衡量算法性能的标杆。
FIFO	置换时将最早被调入的页面调出内存。	实现简单，但性能较差且存在 Belady 异常。
LRU	置换时将最长时间未被使用的页面置换出内存。	性能较好，接近 OPT。但是实现复杂，需要特定的硬件支持。
CLOCK	循环扫描页面，将首个访问位 = 0 的页面置换出内存。遇见访问位 = 1 的页面时，将其访问位设置为 0。	实现简单，效果略差于 LRU。 未考虑页面是否修改， 写磁盘次数多于改进型 CLOCK。
改进型 CLOCK	将访问位和修改位对记为 (P, M)，取值 0 表示未访问或未修改，取值 1 表示已访问或已修改。此方法按 (0, 0)、(0, 1)、(1, 0)、(1, 1) 的优先次序置换页面。	实现简单，效果好于 CLOCK 但依旧略差于 LRU。

【提示】考生学到此处时，可先完成本节末习题精编部分的第 01 ~ 24 题，以巩固相关知识。

3.2.5 内存映射文件

虚拟存储器解决了内存空间不足的问题，但这一机制同时也会给系统带来较多的 I/O 开销。如图3.31所示，若能使用**内存映射文件**（memory-mapped file）则可以对这一机制做进一步的完善。内存映射文件是指进程通过发起一个系统调用，实现从进程对应磁盘文件到其虚拟内存空间中某部分的映射。这一机制可以提升系统的 I/O 效率，从而减少虚拟存储器带来的 I/O 开销。

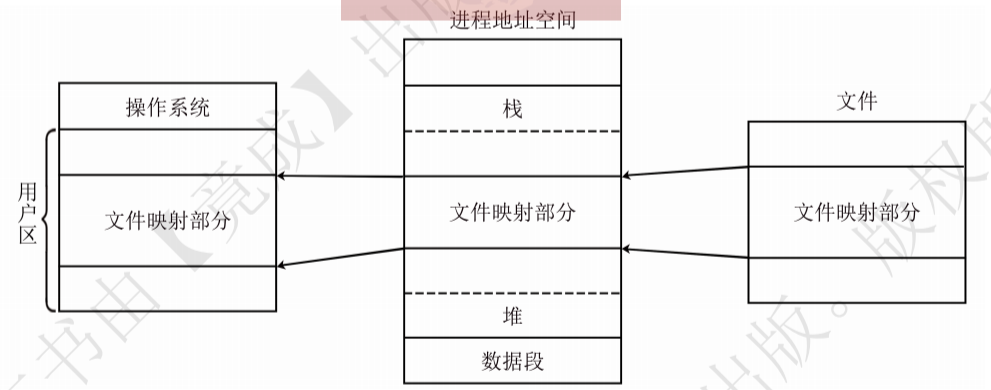

图 3.31　内存映射文件示意图

【提示】想要了解内存映射文件为什么可以带来系统 I/O 效率的提升，考生需要了解设备管理部分关于 I/O 系统中内存交换缓冲区的内容。内存映射文件这部分知识并不影响考生接下来的学习，若考生暂时未能理解，大可先略过此节，待学习过内存交换缓冲区的内容后再学习此节内容。

现在先对基本的 I/O 过程做一个简单介绍，I/O 指计算机主机与各种设备之间的通信过程，而各种外部设备的读取速度差距巨大，并且大多都与 CPU 的处理速度存在巨大的差距。为了缓解这一速度矛盾，系统在内存中开辟一块连续区域作为交换缓冲区，这块区域属于操作系统的内核部分。主机与设备之间需要传输的数据都统一先暂存到这片交换缓冲区中，待具有一定规模后再统一写出到设备或写入到用户区。