

## 操作系统 第九次作业

姓名：朱首赫

学号：2023K8009906029

9.1 假设一台计算机上运行的一个进程其地址空间有 8 个虚页（每个虚页大小 4KB，页号 1 至 8），操作系统给该进程分配了 4 个物理页框（每个页框大小为 4KB），该进程对地址空间中虚页的访问顺序为 1 2 3 5 4 3 7 3 7 8 6 1。假设分配给进程的 4 个物理页框初始为空，请计算：

(1) 如果操作系统采用 CLOCK 算法管理内存，那么该进程访存时会发生多少次 page fault？当进程访问完上述虚页后，物理页框中保存的是哪些虚页？

(2) 如果操作系统采用 LRU 算法管理内存，请再次回答 (1) 中的两个问题。请回答虚页保存情况时，写出 LRU 链的组成，标明 LRU 端和 MRU 端。

解：(1) 如图 1 所示，共发生 9 次 page fault，最终物理页框中保存的虚页为 1、6、7、8。

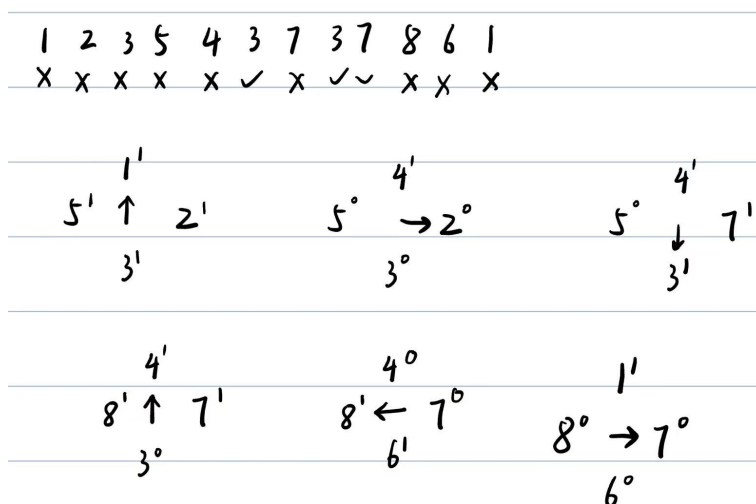


图 1: CLOCK 算法内存管理过程

(2) 如图 2 所示，共发生 9 次 page fault，最终物理页框中保存的虚页为 1、6、7、8。

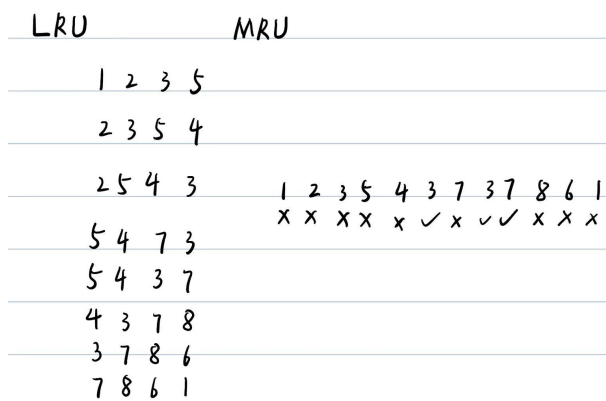


图 2: LRU 算法内存管理过程

**9.2** 假设一台计算机给每个进程都分配 4 个物理页框，每个页框大小为 1KB。现有一个程序对一个二维整数数组（`uint64 X[32][32]`）进行赋值操作，该程序的代码段占用一个固定的页框，并一直存储在内存中。程序使用剩余 3 个物理页框存储数据。该程序操作的数组 X 以列存储形式保存在磁盘上，即 `X[0][0]` 后保存的是 `X[1][0]`、`X[2][0]`...`X[31][0]`，然后再保存 `X[0][1]`，以此类推。当程序要赋值时，如果所赋值的数组元素不在内存中，则会触发 page fault，操作系统将相应元素以页框粒度交换至内存。如果该进程的物理页框已经用满，则会进行页换出。该程序有如下两种写法。

写法 1:

```
for(int i=0;i<32;i++)
    for(int j=0;j<32;j++)
        X[i][j] = 0
```

写法 2:

```
for(int j=0;j<32;j++)
    for(int i=0;i<32;i++)
        X[i][j] = 0
```

请分析使用这两种写法时，各自会产生多少次 page fault?（注：请写出分析或计算过程）

**解：**每个物理页框能够存储  $1\text{KB}/64\text{bit} = 128$  个数组元素，即 4 列数据。存储所有数组元素需要 8 个物理页框。假设使用 LRU 内存管理算法。

写法 1 中，内存循环连续访问不同列的数组元素，那么每 4 次访问就会跨页，触发一次 page fault。每完成一次外层循环，内存中留下的都是后 16 列的元素，所以两次外层循环间的访存操作仍然会触发 page fault。再考虑到第一次的 cold miss，因此 page fault 次数 =  $32 \times 32/4 - 1 + 1 = 256$  次。

写法 2 中，内存循环连续访问同列的数组元素，故内层循环操作之间不发生 page fault。那么除第一次的 cold miss，只有在每 4 次外层循环间才会触发一次 page fault，因此 page fault 次数 =  $32/4 - 1 + 1 = 8$  次。

**9.3** 假设一个程序有两个段，其中段 0 保存代码指令，段 1 保存读写的数据。段 0 的权限是可读可执行，段 1 的权限是可读可写，如下所示。该程序运行的内存系统提供的虚址空间为 14-bit 空间，其中低 10-bit 为页内偏移，高 4-bit 为页号。

当有如下的访存操作时，请给出每个操作的实际访存物理地址或是产生的异常类型（例如缺页异常、权限异常等）

- (1) 读取段 1 中 page 1 的 offset 为 3 的地址
- (2) 向段 0 中 page 0 的 offset 为 16 的地址写入
- (3) 读取段 1 中 page 4 的 offset 为 28 的地址
- (4) 跳转至段 1 中 page 3 的 offset 为 32 的地址

Segment 0		Segment 1	
Read/Execute		Read/Write	
Virtual Page #	Page frame #	Virtual Page #	Page frame #
0	2	0	On Disk
1	On Disk	1	14
2	11	2	9
3	5	3	6
4	On Disk	4	On Disk
5	On Disk	5	13
6	4	6	8
7	3	7	12

图 3

解：(1) 0x1110,0000000003

(2) 写权限异常

(3) 缺页异常

(4) 执行权限异常

**9.4** 假设一个程序对其地址空间中虚页的访问序列为 0,1,2, ...,511,422,0,1,2,...,511, 333,0,1,2, ..., 即访问一串连续地址（页 0 到页 511）后会随机访问一个页（页 422 或页 333），且这个访问模式会一直重复。请分析说明：

(1) 假设操作系统分配给该程序的物理页框为 500 个，那么，LRU, Clock 和 FIFO 这三种算法中哪一个会表现较好（即提供较高的命中率），或者这三种算法都表现不佳？为什么？

解：下面分别计算三种算法在不计 cold miss 且程序无限执行该访问模式时的命中率。

LRU 在第一段连续访问中会一直无法命中，在即将访问随机页时，内存中保留的虚页为 12-511。(1) 若命中 422，保留虚页 12-421、423-511、422；在下一段连续访问中，即将访问到 422 时，内存中保留的虚页为 435-511、422、0-421，访问后保留虚页 435-511、0-421、422；即将访问随机页时，内存中保留的虚页又回到 12-511 (2) 若命中 333，和上述情况同理，只保证在下段连续访问中命中一次。因此 LRU 命中率 =  $2/513 \approx 0.39\%$

Clock 算法中，由于命中页会多出一保留机会，所以它和 LRU 表现一致，因此 Clock 命中率 =  $2/513 \approx 0.39\%$

FIFO 算法中，没有第二次机会，只有随机访问时可以命中一次，因此 FIFO 命中率 =  $1/513 \approx 0.19\%$

综上所述，在题目设条件下，LRU 和 Clock 的表现略好于 FIFO，但实际表现都极其不佳，几乎无法命中。这是因为这种大跨度的循环扫描破坏了缓存机制所依赖的局部性原理假设，任何利用局部性原理的算法都不会有好的表现。

因为扫描工作流大于缓存大小