## 作业 10

10.1 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（每个虚页大小为 4KB，页号为 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 3 4 6 2 3 5 4 7 8。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。

解答：

CLOCK 算法是一种近似于 LRU 的页面替换算法，它通过维护一个循环队列和一个指针来模拟 LRU 算法。当需要替换页面时，它会检查指针指向的页面的访问位，如果访问位为 0，那么就替换这个页面，否则就将访问位设置为 0 并将指针向前移动，直到找到一个可以被替换的页面。

LRU 算法是一种页面替换算法，它总是替换最长时间未被访问的页面。在实际操作中，通常使用一个链表来实现 LRU 算法，链表的头部是最近使用的页面，尾部是最长时间未使用的页面。

（1）对于 CLOCK 算法：假设如果物理页框全都为空，那么当新的页面被加载到物理页框中时，指针会移动到新加载的页面的下一个位置，也就是说此时加载进入的物理页访问位均为 1.

访问序列为：1 3 4 6 2 3 5 4 7 8

初始状态，物理页框为空，所以前 4 个访问（1 3 4 6）都会导致页面错误，此时物理页框为：1 3 4 6，指针指向物理页 1，当物理页 6 加载后指针指向下一页，即循环指向 1.

访问 2，页面错误，1 3 4 6 访问位被依次置 0 后，最终替换 1，物理页框为：2 3 4 6，访问位依次为 1 0 0 0，指针指向 3

访问 3，没有页面错误，物理页框为：2 3 4 6，访问位依次为 1 1 0 0，指针指向 4

访问 5，页面错误，替换 4，物理页框为：2 3 5 6，访问位依次为 1 1 1 0，指针指向 6

访问 4，页面错误，替换 6，物理页框为：2 3 5 4，访问位依次为 1 1 1 1，指针指向 2

访问 7，页面错误，同第一次访问位均为 1，依次置为 0，最后替换 2，物理页框为：7 3 5 4，访问位依次为 1 0 0 0，指针指向 3

访问 8，页面错误，替换 3，物理页框为：7 8 5 4，访问位依次为 1 1 0 0，指针指向 5

所以，总共有 9 次页面错误，最后物理页框中保存的虚页为：7 8 5 4，指针指向 5

（2）对于 LRU 算法：

访问序列为：1 3 4 6 2 3 5 4 7 8

初始状态，物理页框为空，所以前 4 个访问（1 3 4 6）都会导致页面错误，此时物理页框为：1 3 4 6，LRU 链为：1 3 4 6，LRU 端为 1，MRU 端为 6

访问 2，页面错误，替换 1，物理页框为：2 3 4 6，LRU 链为：3 4 6 2，LRU 端为 3，MRU 端为 2

访问 3，没有页面错误，物理页框为：2 3 4 6，LRU 链为：4 6 2 3，LRU 端为 4，MRU 端为 6

访问 5，页面错误，替换 4，物理页框为：2 3 5 6，LRU 链为：6 2 3 5，LRU 端为 6，MRU 端为 5

访问 4，页面错误，替换 6，物理页框为：2 3 5 4，LRU 链为：2 3 5 4，LRU 端为 2，MRU 端为 3

访问 7，页面错误，替换 2，物理页框为：7 3 5 4，LRU 链为：3 5 4 7，LRU 端为 3，MRU 端为 7

访问 8，页面错误，替换 3，物理页框为：7 8 5 4，LRU 链为：5 4 7 8，LRU 端为 5，MRU 端为 8

所以，总共有 9 次页面错误，最后物理页框中保存的虚页为：7 8 5 4，LRU 链为：5 4 7 8，LRU 端为 5，MRU 端为 8

10.2 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 512B。现有一个程序对一个二维整数数组（uint32 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;i++)
   for(int i=0;i<32;j++)
      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）

解答：

数组以列存储形式保存在磁盘上，表明 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。现在每个页框大小为 512B，而每个 uint32 类型的整数占用 4B，所以每个页框可以存储 128 个 uint32 整数，也即如果导入一个新页时，会同时导入 128 个数组元素，而这 128 个数组元素等价于发生 4 次行改变。

总共会访问 32*32=1024 个元素，也即八个物理页的内存。

对于写法 1：

这个算法是按行访问数组的，但是数组是按列存储的，所以如上述所提到，每访问 4 个不同的行时就会发生一次页替换。

由于物理页框总数为 4，假设采用的是 FIFO 算法或者其他保证替换页公平的换页算法，其中一个页存储的是代码段数据，只有三个物理页可以用于进行页替换。数组访问某一个页后发生替换，每八次页错误会回到同一个页，那么

对于只有三个页来说，不能同时放下 8 个页的数据，必然会在外部每次循环中发生 8 次页错误，总共也就是 8*32=256 次也错误。

对于写法 2：

此时是按列访问数组的，与数组的存储方式相同，所以每访问 128 个元素时，才会触发 page fault。总共有 1024 个元素，写法 2 只会产生 1024/128=8 次 page fault。

10.3 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。

| 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 |

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

解答：题目中没有给出物理页框的大小，但是一般而言是跟虚拟页大小一致，这里假设其为 2^10 也即 1kb 的页面大小。虚拟地址到物理地址的转换对于该题来说可以看作是一级页表，那么即在 page_frame 中找到 PPN 号，再加上偏移即可。

（1）、段 1 中 page 1 对应于第 14 个物理页框，同时段 1 的权限为可读可写，读操作不会发生异常。物理地址为 0x3800+3=0x3803；(14kb+3)

（2） 、段 0 中 page 0 已经对应于第 2 个物理页框，段 0 的权限为可读可执行，写操作会触发权限异常。物理地址为 0x800+0x10=0x810；

（3）、段 1 中 page 4 存储在磁盘上，会触发缺页异常。起始地址取决于后续其 load 进内存的物理块号，设之十六进制为 0xM，则实地址 0xM*0x400+0x1b；

（4）、段 1 中 page 3 对应第 6 个物理页框，但段 1 的权限为可读可写，执行操作会触发权限异常。其物理地址为 0x1800+0x20=0x1820.

10.4 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，Second Chance 和 FIFO 这三种算法中哪一个会表现较好（即提供较高的缓存命中率），或是这三种算法都表现不佳？为什么？

LRU：由于程序首先访问页 0 到页 511，这些页将按照访问顺序排列在 LRU 队列中依次添加到队尾，即 MRU 端上。当程序随机访问一个页时（如 422、333 等），这些页将被移动到 LRU 队列的 MRU 端，而页 0（如果页 0 未被访问到）仍放置在队列的开头，如果页 0 被访问到，LRU 端则为页 1，以此类推。

当程序从页 0 开始访问时，由于操作系统为该程序分配了 500 个物理页框，因此页 0 到页 499 将仍然在内存中，从而产生命中。只有当访问到页 500 时，才会发生缺页中断，此时在 LRU 端的页将被替换出去，新替换进来的页将位于 MRU 端。这种情况下，第二轮循环开始（从第一次访问开始每访问 513 页，包括随机访问的页），除了第一轮随机访问过的页会被置于队尾从而发生命中，其他的页均会被替换过一次，最后队列中的结果和第一次循环结束后的结果大致相同，除了被第一次随机访问扰乱的某一页不同，第二次随机访问时，页依然可能命中或者不命中，命中的几率会更大，不命中的话，下次访问的头几页将会有一页不需要被替换。

依次类推，假设一共进行了 N 轮循环，不考虑第一次导入 500 页发生的缺页中断，随着循环次数的增大，最后随机命中的效果几乎完全取决于随机访问那次产生的命中，那么命中率差不多为 只有（1/512）.

Second Chance：该算法维护了一个循环队列，当一个页面首次被加载到内存时，它会被放到队列的尾部。当这个页面再次被访问时，它的引用位会被设置为 1，但它的位置不会改变。当需要替换页面时，Second Chance 算法会查看队列头部的页面。如果这个页面的引用位为 0，那么它会被替换掉；如果引用位为 1，那么这个页面会被放到队列的尾部，引用位被设置为 0，然后算法会继续查看下一个页面。（这一个过程通常是在时钟中断的过程中进行的，也就是说如果一个页在一段时间内没有被访问过，这个有效位就会被置为 0）

考虑到以上前提，最开始开始访问的 500 页导入之后，500-511 页会将最开始 0-11 页替换出去,最后移到队列的末尾，有效位被置为 0，或者位于最开始 12 页中，但有效位是为 1. 当进行随机访问的时候，依然是命中的几率会更高，如果命中其中的某页，该页访问位会被置位 1 也就是具有 Second Chance，它的位置不会改变，如果不命中，将会替换队列开头的那页有效位为 0 的页。所以这两这种算法的行为是基本一致的，最后命中率也差不多一致，完全取决于最那次随机访问产生效果。

FIFO：对于 FIFO 算法，当程序开始第二轮访问页 0 到页 511 时，FIFO 将替换掉最早加载到内存中的页，随机范访问时，命中率依然很高，但是不命中的时候会直接替换在队头的页，而命中的时候也不会进行置一的操作。这将导致每次循环时都会发生缺页中断，结果只会由随机访问时是 0-11 页还是 12-511 页发生变动，如果随机访问的结果是不命中，那么会产生扰动，否则不会对结果产生影响，也就是说命中率的效果可以算是前两种的 12/512 左右.

总的来说，这种这三种算法都表现不佳，因为在页面访问模式是在随机且无规律的情况下，尽管可以预测每 500 页的访问情况，但是不能预测随机访问时产生的效果如何。但是 FIFO 算法，因为它无法考虑到页面的使用频率，可能导致最近被访问的页面被置换出去，从而影响缓存命中率，所以对比来说 FIFO 算法依然是效率最低的。