操作系统课程项目二

——请求调页存储管理方式模拟

2051498 储岱泽

一、项目目的

通过此次项目的实践，能够学习调页存储管理中页面、页表、地址的转换，了解页面置换的过程，从而加深对请求调页系统的原理和实现过程的理解。

二、开发环境

- 操作系统平台：MacOS
- 开发软件：VScode , Chrome 浏览器
- 开发语言：Html5 , CSS3 , javascript

三、项目需求

- **基本任务：**

假设每个页面可存放 10 条指令，分配给一个作业的内存块为 4。模拟一个作业的执行过程，该作业有 320 条指令，即它的地址空间为 32 页，目前所有页还没有调入内存。

- **模拟过程：**

在模拟过程中，如果所访问指令在内存中，则显示其物理地址，并转到下一条指令；如果没有在内存中，则发生缺页，此时需要记录**缺页次数**，并将其调入内存。如果 4 个内存块中已装入作业，则需进行页面置换。所有 320 条指令执行完成后，计算并显示作业执行过程中发生的**缺页率。**

- **置换算法：**

1.采用先进先出(FIFO)置换算法。
2.最近最久未使用（LRU）算法

- **作业中指令访问次序原则：**

50%的指令是顺序执行的，25%是均匀分布在前地址部分，25%是均匀分布在后地址部分。

- **具体实施方法：**

- 在 0－319 条指令之间，随机选取一个起始执行指令，如序号为 m。
- 顺序执行下一条指令，即序号为 m+1 的指令。
- 通过随机数，跳转到前地址部分 0－m-1 中的某个指令处，其序号为 m1。
- 顺序执行下一条指令，即序号为 m1+1 的指令。
- 通过随机数，跳转到后地址部分 m1+2~319 中的某条指令处，其序号为 m2。
- 顺序执行下一条指令，即 m2+1 处的指令。
- 重复跳转到前地址部分、顺序执行、跳转到后地址部分、顺序执行的过程，直到执行完 320 条指令。

四、算法分析

- **FIFO 算法介绍**

FIFO 算法，也被称为先进先出算法，是操作系统中用于管理内存中页面置换的一种算法。这个算法的思想很简单：当页面需要被替换时，选择最早进入内存的页面进行置换。即，先进入内存的页面先被置换掉，而最近进入内存的页面则保持在内存中。

具体实现中，操作系统通常维护一个队列，用于按照页表中的顺序记录内存中的页面。当新的页面进入内存时，它会被添加到队列的末尾。当需要替换页面时，操作系统选择队列中的第一个页面进行置换。如果该页面被修改，则需要将其写回磁盘，以确保数据不会丢失。

优点：实现简单、易于理解。
缺点：性能较差，容易产生抖动现象。

- **LRU 算法介绍**

LRU 算法，即最近最少使用算法，是用于管理操作系统内存中页面置换的一种算法。它的基本思想是当页面需要被替换时，将最近最久未被使用的页面替换出去。

具体实现中，我们可以记录下每个页面上一次被访问的最后时间，以后每执行一次指令如果发生缺页现象，就将最近访问频率最少，上一次访问时间最早的那个页面进行替换。

优点：利用了程序局部性原理，性能较好，最接近“最佳算法 OPT”。
缺点：需要额外空间记录每个页面被访问的最后时间，且实现比较复杂。

五、算法实现

- **FIFO 算法的实现**

首先，系统会先给 4 个内存块初始化，随机装入 4 条指令。如果用户选择了按照 FIFO 算法来进行调页存储管理，则首先系统选择指令的访问次序是“顺序执行”、“向后跳转”还是“向前跳转”。其中，“顺序执行”表示按照指令序号逐个执行，而“向后跳转”和“向前跳转”则表示随机跳转到某个指令。

选择了指令的执行次序之后，系统根据选择的策略，生成接下来要执行的指令。

在执行每个指令时，首先判断该指令是否被执行过。如果没有被执行过，则判断该指令是否在内存中。如果在内存中，则直接执行该指令；反之，如果缺页，则将该指令所在的页调入内存，并**替换掉 FIFO 算法中最先进入内存的页**。执行完一个指令后，将该指令标记为已执行，并更新界面上显示当前缺页个数和缺页率的标签。

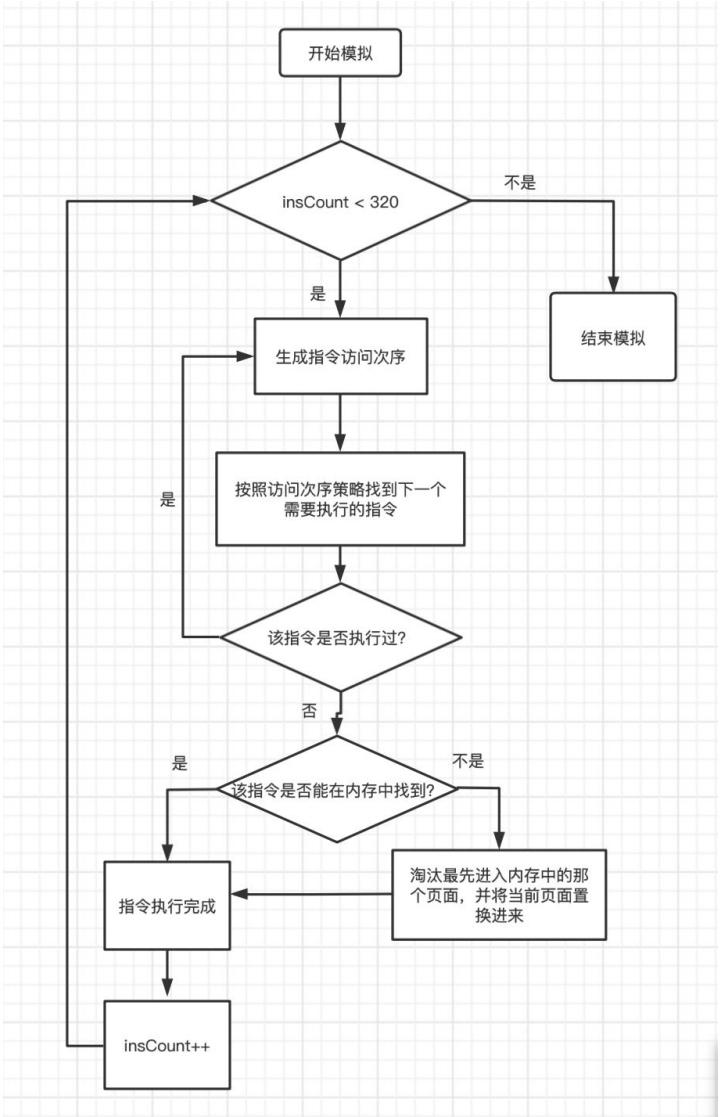


图 1 FIFO 算法流程图

- **LRU 算法的实现**

同样，在用户选择了 LRU 算法来进行页面的置换，并且按下开始按钮之后，系统先为 4 个内存块分配 4 个指令。然后对于接下来生成的需要装载执行的指令，系统首先判断该指令是否执行过，如果没有执行过，再判断在内存中是否能够找到该指令的页，如果不能找到考虑按照 LRU 规则进行页面置换。

按照 LRU 规则进行置换要求我们找到内存中最久没有被访问过的页面。那我们如何找到这样的页面呢?

我们可以引进一个 stack 数组来进行这样的记录。stack 数组一共有 4 个元素，其中越往前的元素表示越久没有被访问的，越往后的元素表示越最近被访问的。我们每访问一个页我们就将该页放到 stack 数组的末尾；每次需要发生页面置换的时候，我们就把 stack 的第一位所代表的内存块中的内容置换掉。

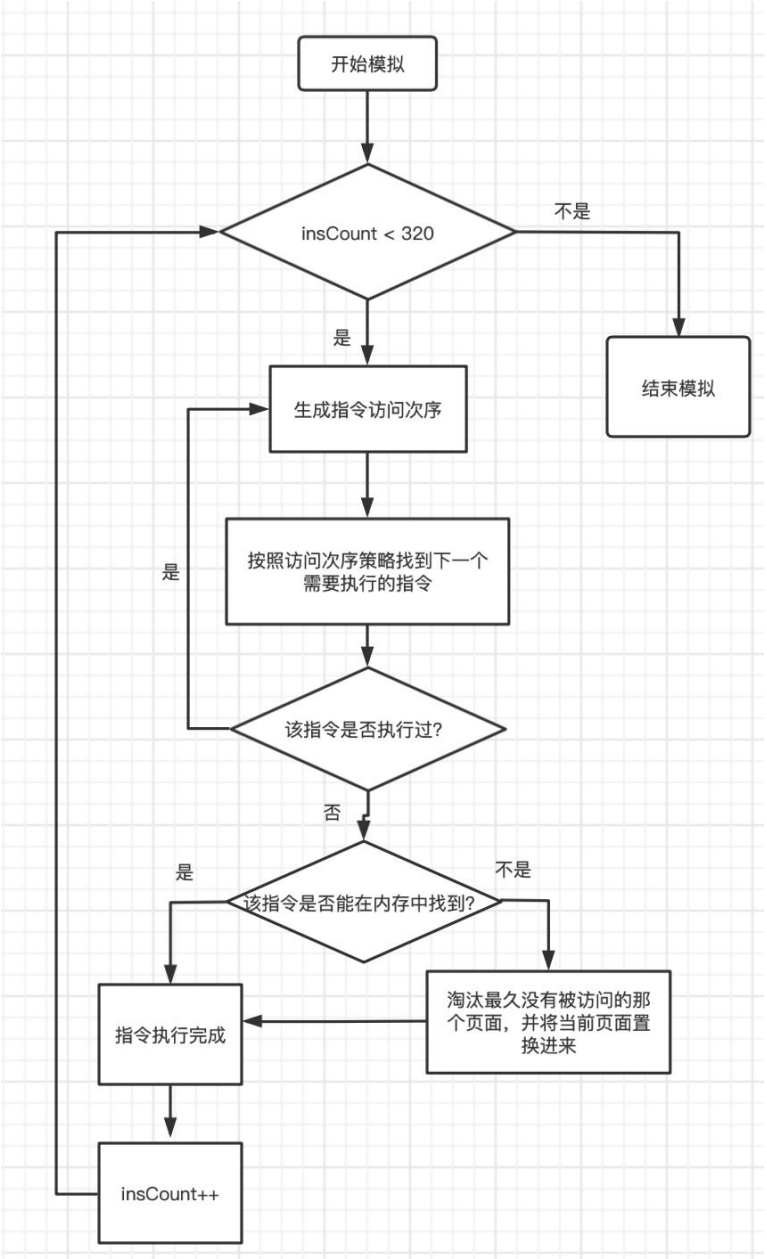


图 2 URL 算法流程图

六、项目整体逻辑

- 项目主要函数列表

	函数名称	函数功能
1	**isInstructionExecuted**(number)	判断指令是否已经被执行过
2	**isInstructionAvailable**(number)	判断内存块中是否有该页面
3	**init**( )	初始化
4	**initMemory**( )	初始化内存状态
5	**FIFO**( )	FIFO 算法的具体实现过程
6	**LRU**( )	LRU 算法的具体实现过程
7	**chooseAlgorithm**( )	读取用户选择的算法
8	**start**( )	开始模拟

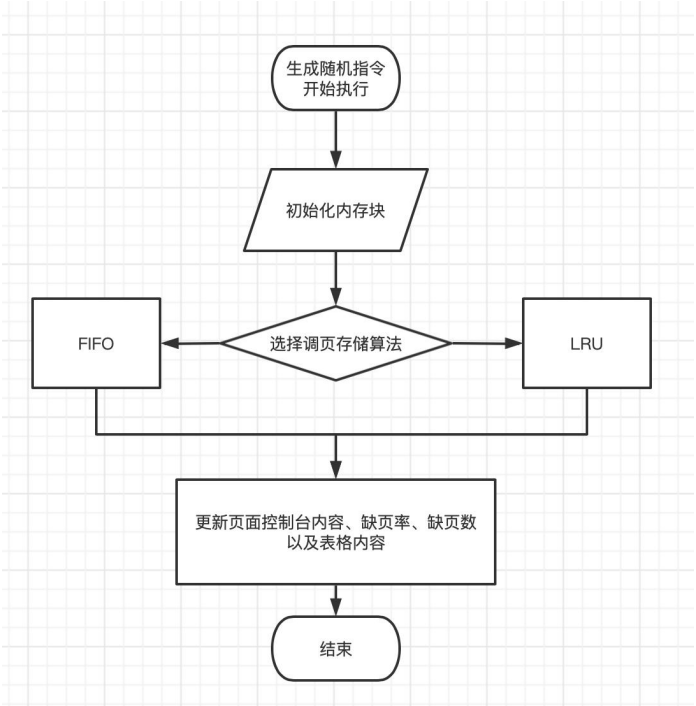
- 项目主要流程

用户进入页面后首先应该先通过单选按钮选择希望运行的算法，然后按下屏幕中央的“开始”按钮，接下来触发页面监听，并且调用相应的函数 start( )，开始模拟相应的调页存储方式。

首先，在 start( )函数当中调用 init( )函数，然后调用 initMemory( )函数进行内存的初始化，给每个内存块先随机分配一个指令，然后开始调用 FIFO( )或者 LRU( )函数进行相应的算法的处理。

在判断是否需要页面置换之前，先随机生成一个指令，利用 isInstructionExecuted( )判断该指令是否已经被执行过了，如果执行过了，就重新生成一个指令；如果没有就用 isInstructionAvailable( )判断其是否已经在内存中出现过。如果出现过，那就直接执行；如果没有，就需要按照用户选择的算法进行页面的置换，并将缺页数加一，重新计算更新缺页率。

接下来判断是否已经执行完 320 条指令，如果执行完了，就结束；没有就继续生成新指令，重复以上步骤。



七、界面设计

当用户打开 html 文件可以看见如下所示的界面，该界面提供了两个单选按钮供用户选择想要模拟的算法。同时中间有四个圆角矩形，分别代表四个内存块。在 4 个内存块的中间有一个“开始”按钮，按下该按钮内存管理模拟开始执行。

除此之外，界面的右侧还有一个侧边栏，在侧边栏中，用户可以查看当前指令、缺页数以及缺页率等信息，用户也可以在控制台中看见运行过程输出的信息。最底下有对算法的介绍。

按下“开始运行”的按钮之后算法开始执行，同时可以看见每个内存块当前的指令会显示其中，右边侧边栏内也可以看见具体详细的运行信息以及缺页数和缺页率。

除此之外，在四个内存快下面的表格中也会详细记录每一次发出的缺页中断，以及每个内存块内容变化的过程。