**请求调页存储管理方式模拟设计方案**

2253893 苗君文

**1. 项目简介**

**1.1. 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。

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。

**1.2. 功能描述**

本内存管理项目模拟请求调页存储管理方式，支持FIFO（先进先出）和LRU（最近最久未使用）两种页面置换算法。用户可以通过该模拟器观察内存页面置换的过程，并查看缺页次数和缺页率。用户也和可以查看内存块的变化，包括指令代号、内存块状态、是否缺页、放入内存块号和换出页号等信息。

**2. 开发环境**

系统 ：Windows 11

IDE ：Visual Studio Code

HTML 版本 ：HTML5

CSS 版本 ：CSS3

JavaScript 版本 ：ES6

依赖：jQuery3.6.0

**3. 系统架构**

**3.1. 项目结构**

该内存管理项目使用HTML、CSS和JavaScript构建。项目结构如下：

.

├── index.html

├── style.css

├── script.js

└── README.md

index.html：定义了页面的布局和结构，包括页面标题、表格以及按钮等元素。

style.css：定义了页面的样式，包括表格、按钮的样式和特定单元格的颜色等。

script.js：包含了项目的主要逻辑，包括页面置换算法的实现、指令生成、页面更新等功能。

**3.2. 类设计**

该项目使用JavaScript类来模拟内存页和内存管理。

**3.2.1. Page 类**

Page 类用于表示内存中的一个页。

属性：

id：页的唯一标识符。

instructions：包含的指令集合，每个页包含10条指令。

**3.2.2. Memory 类**

Memory 类用于表示内存，负责管理内存中页的存取和置换。

属性：

size：内存的总页数。

method：使用的置换算法。

frames：当前在内存中的页。

lastUsedTime：记录每个页上次使用的时间，用于LRU算法。

相关函数：

setMethod(method)：设置置换算法。

hasInstruction(id, time)：检查内存中是否有指定的指令，并根据算法更新页的使用时间。

**4. 核心算法**

本项目模拟的请求调页存储管理方式是一种动态内存管理技术，用于在计算机系统中实现虚拟内存管理。它通过在需要时将页面从磁盘加载到内存中，从而高效地管理内存资源。本项目使用的置换算法有先进先出（FIFO）和最近最久未使用（LRU）。

请求调页存储管理方式需要注意以下三个概念：

页表：操作系统维护一个页表，记录每个虚拟页面对应的物理内存地址。如果页面在内存中，页表包含该页面的物理地址；如果页面不在内存中，页表指示页面在磁盘上的位置。

页面置换：当需要访问的页面不在内存中（缺页），操作系统从磁盘中加载该页面到内存。如果内存已满，则需要选择一个页面置换出去（换页）。FIFO算法移除队列中最早进入内存的页面，并将新页面加载到该位置。LRU选择最后使用时间最早的页面进行置换，将新页面加载到该位置，并更新其最后使用时间。

缺页中断：当程序访问的页面不在内存中时，会发生缺页中断。操作系统处理缺页中断，通过页面置换算法将所需页面加载到内存。

**4.1. FIFO页面置换算法**

（1）初始化：

创建一个包含32个页面对象的数组，每个页面对象表示内存中的一个页面。这些页面包含唯一标识符和一组指令。

创建一个容量为4的内存对象，表示可容纳4个页面的内存空间。

将内存对象的置换算法设置为FIFO。

（2）指令生成：

生成一组模拟程序执行的指令序列，这些指令将用于访问内存页面。

（3）访问页面：

遍历生成的指令序列，每访问一条指令，计数器（visit）加一，表示当前的访问次数。

检查当前访问的指令是否在内存中。如果指令在内存中，表示页面命中，将相关信息记录到内存表格中，并继续访问下一条指令。

（4）缺页处理：

如果当前访问的指令不在内存中，表示发生了缺页（页面不在内存中）。计数器（totalMiss）加一，表示增加了一次缺页。

检查内存是否已满。如果内存未满，直接将页面加载到内存中。如果内存已满，则需要选择一个页面进行置换。

根据FIFO策略，选择内存中最早进入的页面进行置换。将当前需要访问的页面替换掉该页面。

更新被置换页面的记录，记录下被置换页面的标识符。

将当前访问的页面加载到内存中，更新该页面的最后访问时间。

无论缺页与否，都可以通过循环计数器block来找到要放入的内存块位置，因为根据FIFO，所以设计了循环访问内存块的方式，在有内存空间时block指向的就是可放的空内存块，而内存空间已满时，需要找最早放入的页，由于是循环顺序放入页，所以block当前指向的就是最早放入的页所在的内存块。

（5）循环处理：

重复上述过程，直至遍历完所有的指令序列。

（6）计算缺页率：

遍历完所有指令后，计算总的缺页率。缺页率等于缺页次数除以指令总数。

**4.2. LRU页面置换算法**

其思路与4.1.FIFO算法类似，仅缺页处理部分有区别。下面将介绍缺页处理部分：

如果当前访问的指令不在内存中，发生缺页。缺页次数（totalMiss）加一。

首先检查内存中是否还有空闲的内存块。如果有空闲内存块，则将页面直接加载到空闲内存块中：查找内存中第一个未使用的内存块（即内存块数组中值为undefined的位置）。

将页面加载到这个空闲的内存块中，并更新这个内存块的最后访问时间。记录该内存块的编号。如果内存已满，没有空闲内存块，则需要置换一个页面：

遍历内存中的所有内存块，找到最久未使用的内存块。具体步骤是：比较所有内存块的最后访问时间，找到具有最小最后访问时间的内存块。记录被置换页面的标识符，并将其从内存中移除。将新的页面加载到被置换的内存块中，并更新该内存块的最后访问时间。

记录该内存块的编号和被置换页面的标识符。

每次指令访问后，无论是否发生缺页，都将当前内存状态和访问结果记录到内存表格中。内存表格展示当前访问的指令编号、内存中页面的状态、是否缺页、插入的内存块编号以及被置换的页面标识符。

**4.3. 生成指令**

生成指令的算法通过generateInstructions函数实现。

首先创建一个包含320个指令编号的数组，并初始化一个用于记录生成指令数的计数器以及一个用于存放指令序列的空数组。通过循环不断生成指令，直到生成320条指令。

在每次循环中，算法会先生成一个随机的指令编号m，并将其添加到指令序列中。如果生成的指令编号小于319，则添加其相邻的下一个指令m+1。接着，生成一个在0到m-1范围内的随机前地址指令m1，并添加到指令序列中，如果m1小于m-1，则添加m1的下一个指令m1+1。然后，生成一个在m+2到319范围内的随机后地址指令m2，并添加到指令序列中，如果m2小于319，则添加m2的下一个指令m2+1。

这样确保了生成的指令序列中不仅有随机指令，还包括相邻指令和地址跳转指令，模拟了实际程序中的指令访问模式。最后，返回生成的完整指令序列。

**5. 用户界面设计**

项目的用户界面主要包括三个部分：参数信息表、页面置换算法控制面板和内存状态表。这些部分通过布局使得用户能够方便地查看和操作。

用户可以点击页面置换算法表的绿色按钮触发FIFO算法或LRU算法，而后生成结果，这是由jQuery事件处理实现的。项目使用jQuery监听页面加载完成后的事件。为FIFO和LRU按钮设置点击事件处理器，用户点击按钮时，分别调用fifo ()和lru ()函数来运行对应的页面置换算法模拟。点击按钮以后，按钮上的文字会从“开始”变为“重启”。

内存状态表中，项目也构建了让用户更清晰看出结果的设计。当访问的指令不在内存中，需要将其加载到内存，此时发生缺页，此行将会显示为蓝色。当访问的指令已经存在于内存中，不需要进行页面置换，此行将会显示为白色。当内存块有变换，即置换或装入时，单元格颜色将呈现为黄色。

初始界面：



运行后界面：



**6. 心得体会**

**6.1. 对请求调页存储管理方式的理解**

通过实现FIFO和LRU两种页面置换算法，我对请求调页存储管理方式有了更深入的理解。我学会了如何根据不同的算法逻辑来实现页面置换，以及通过缺页次数和缺页率评估算法的效果。在模拟内存管理过程中，我体会到了页面置换对系统性能和用户体验的影响。特别是LRU算法的实现过程中，需要维护额外数据结构的复杂性，但也领悟到了其对系统优化的重要性。FIFO算法虽然实现逻辑简单，但是会出现抖动现象，性能较差。而LRU算法基于程序的局部性原理，其性能是最接近OPT最佳算法的置换算法。

**6.2. 对前端开发的学习与应用**

作为初学者，我通过这个项目初步掌握了前端开发的基本知识和技能，初步能应用HTML、CSS和JavaScript等前端开发技术。我学会了如何使用HTML构建页面结构，通过CSS美化页面样式，以及通过JavaScript实现页面交互和动态效果。我体验到了这三种技术的紧密配合，以及它们对于网页功能实现的重要性。并且在程序运行不成功时，可以通过浏览器中按F12，查看控制台进行debug，我也学会使用console.log语句来进行调试查看。这次项目的学习对我未来的开发有很大的帮助。