Tongji University

操作系统课程项目实验报告

002-请求分页分配方式模拟

专　　业：软　件　工　程

指导教师：张　惠　娟

学　　号：２１５３０６１

姓　　名：谢　嘉　麒

1. 项目及基本要求概述

1.1.项目概览

1.1.1. 项目目的

（1）学习页面、页表、地址转换相关知识。

（2）学习页面置换过程

（3）加深对请求调页系统的原理和实现过程的理解。

1.1.2. 项目要求

（1）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。

（2）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。

（3）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。

（4）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。

（5）置换算法可以选用 FIFO 或者 LRU 算法

（6）作业中指令访问次序可以按照下面原则形成：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。

1.2.项目环境

1.2.1. 编程语言

本次项目需要对接面进行图形化设计，考虑到项目可以模拟为前后端交互的网页形式，同时为了方便插入如算法下拉式选择框等控件，因此选择 web 三件套（即 HTML+CSS+JaveScript）进行实现。

1.2.2. 操作系统平台

本次项目使用 Windows 系统作为操作系统平台，所有程序均在 Windows10 系统下运行。

1.2.3. 项目依赖浏览器

因项目由前端三件套实现，故项目网页依赖浏览器进行呈现。浏览器可以选择 Microsoft

Edge, Google Chrome 或 FireFox 等。

2. 整体设计思路

本次项目整体设计包括四个部分，包括项目界面设计，算法和执行方式选择，结果列表呈现和四个内存块内容分别呈现。

2.1.项目界面设计

项目界面主要包括五个模块：

（1） 项目背景及内容呈现框架：项目背景采用渐变色处理，项目内容主体在框架中呈现。采用 table 标签作为项目主体容器，为 table 设置背景颜色及阴影效果。

（2） 项目主体中首先显示项目名称和提示语句，作为用户交互的提示信息处理。

（3） 主体左侧放置基本信息，包括 Parameter Information 和 Analysis of Results。其中参数信息中标明内存块数量，总指令数和每页存储的指令数。结果分析中标明使用 FIFO 和 LRU 算法计算产生的缺页次数和缺页率。

（4） 主体右侧放置结果列表，包括顺序标号，指令号，四个内存块状态和页状态七列。当用户点击执行按钮后，项目会将每一步的计算结果以列表形式呈现在表格中。

（5） 主体下方呈现四个颜色不同的块。四个块分别表示四个内存块，其中显示当前页号和指令号。单步执行时，页号和指令号随着每一步执行（如果需要改变）而改变，直接执行时呈现结束时的页号和指令号。

2.2.算法和执行方式选择

算法和执行方式的选择放置在主题左侧。算法选择采用下拉式选择框实现。执行方式则根据用户点击“单步执行”或“直接执行”按钮来传入不同参数，进行对应的实现。算法包括 FIFO 和 LRU 两种，在 JS 中分别实现并反馈执行结果和缺页率等参数。

2.3.结果列表呈现

结果列表最初并没有内容，只显示七列标题。当选择“单步执行”时，结果会每间隔 1.5 秒在列表中逐条呈现。当选择“直接执行”时，结果会直接全部呈现在列表中。列表规定固定的占位面积，条数过多时会以右侧滚动条形式进行处理。表格内容关联 JS。

2.4.内存块内容呈现

主体底部四个矩形模块分别代表四个内存块，根据算法每一步执行来获取当前的页号和指令

号，并将其呈现在对应的内存块中。通过将前端标签与 JS 关联，来实现动态的可视化实现。

3. 项目功能实现

3.1.项目界面实现

项目可视化界面由 HTML 和 CSS 渲染得到。页面 HEAD 部分放置同济 logo 以及项目名称。页面 BODY 部分首先设置渐变色背景，并在页面中央放置占位 width:90%;height:90%;的表格。表格整体设置背景和圆角参数。

将头部两行均分别用 colspan = "4"来设置跨列占位，分别提示“Welcome to Memory Arrangement”和项目作者及名称。

表格下一行分为两列，分别设置 colspan = "2"来设置跨列占位。左侧两列合并为一个 table，其中放置四行，分别为参数信息，结果信息，算法选择控件和两个运行按钮。右侧两列合并为一个 table，算法运行前仅设置 thead 呈现，并在表格内容中设置提示语句。算法运行后将隐藏提示语句并显示结果，结果显示方式因点击“单步执行”或“直接执行”不同而不同。

表格最后一行用 colspan = "4"进行跨行占位，在其中设置 table 控件来存放四个内存块，四个内存块占据一行，各分占一列，其中设置“Frame 0x”“页号”和“指令号”。

3.2.FIFO与LRU算法实现

3.2.1. FIFO 算法

FIFO(First in Fist out)算法思想为：当需要淘汰一个页面时，总是选择驻留时间最长的页面淘汰，即先进入主存的页面先淘汰。

FIFO 算法执行思路如下：

（1） 初始化内存块：创建一个大小为 n 的内存块，初始时都为空。

（2） 遍历指令序列：按顺序依次访问指令序列。

（3） 检查页面是否在内存中：对于每一条指令，检查对应的页面是否已经在内存中。

（4） 如果页面已经在内存中，说明命中，继续下一条指令的访问。

（5） 如果页面不在内存中，说明发生缺页，执行下面的步骤：

A. 查找可替换的页面：查找内存块中最先进入内存的页面，即队列中的第一个页面。

B. 替换页面：将最先进入内存的页面替换为当前指令对应的页面，并将新页面放入队列的末尾。

C. 更新缺页计数器：将缺页计数器加 1。

D. 继续下一条指令的访问。

（6） 重复步骤 3 到步骤 6，直到遍历完所有的指令。

FIFO 算法执行函数具体实现如下：

（1） 定义 old 指针来指向最早进入的指令。

（2） 根据要求确定指令执行序列。

（3） 指令不在内存中时发生缺页处理：

A. 有空的内存块则放入内存。

B. 内存块已经占满则替换 old 指向的指令，并更新缺页数量和缺页率。

（4） 更新前端界面显示

3.2.2.LRU 算法

LRU（Least Recently Used）算法思想为：基于最近使用的页面在未来一段时间内可能被再次使用的原则。即将最长时间未被访问的页面替换出去，以便给新的页面腾出空间。

LRU 算法执行思路如下：

（1） 初始化内存：创建一个大小为 n 的内存块，初始时都为空。

（2） 遍历指令序列：按顺序依次访问指令序列。

（3） 检查页面是否在内存中：对于每一条指令，检查对应的页面是否已经在内存中。

（4） 如果页面已经在内存中，说明命中（hit），更新页面的访问时间，并继续下一条指令的访问。

（5） 如果页面不在内存中，说明发生缺页（miss），执行下面的步骤：

A. 查找可替换的页面：在内存中查找最久未被访问的页面。

B. 替换页面：将最久未被访问的页面替换为当前指令对应的页面。

C. 更新页面访问顺序：将新访问的页面移动到内存块的末尾，表示它是最近被使用的。

D. 继续下一条指令的访问。

（6） 重复步骤 3 到步骤 6，直到遍历完所有的指令。

LRU 算法执行函数具体实现如下：

（5） 定义 pos 指针和 record 数组来记录访问时间。

（6） 根据要求确定指令执行序列。

（7） 指令不在内存中时发生缺页处理：

C. 有空的内存块则更新 pos，放入内存。

D. 根据页面访问时间来决定最久未被使用的页面并替换，并更新缺页数量和缺页率。

（8） 更新记录访问时间的 record 数组，将对应页面的访问时间更新为当前的指令计数。

（9） 更新前端界面显示。

3.3.项目关键函数

函数名称	传入参数	返回值	说明
start(mode)	mode，决定单步或直接执行	无	按钮监听函数，点击按钮后调用
IsLeisure()	无	找到的空内存块；未找到则返回-1	寻找空内存块
IsInside(ins)	ins 指令号	已经在内存中则返回-1；否则返回页号	判断指令是否已经在内存中
FindPos()	无	最久未访问的内存块号	寻找最久未访问的内存块号
MakeSequence()	无	无	生成随机指令序列
FIFO(mode)	mode，决定单步或直接执行	无	FIFO 执行算法
LRU(mode)	mode，决定单步或直接执行	无	LRU 执行算法

4. 心得体会

4.1.内存管理项目的认识

通过本次项目，我对内存管理模块有了更进一步的认识，同时也熟悉了FIFO算法和LRU算法的实现方法，通过自主实现来加深了对请求分页分配方式的理解。同时我也在算法实现过程中仔细思考了两种算法各自的优缺点，如下：

4.1.1. FIFO算法优缺点

优点：

（1） FIFO算法非常简单，只需要使用一个队列来维护页面的顺序，使得实现相对容易。

（2） FIFO算法遵循先进先出的原则，保证了每个页面在内存中的停留时间是公平的，不会出现某个页面一直得不到访问的情况。

缺点：

（1）FIFO算法只考虑页面进入内存的时间，而没有考虑页面的访问频率。这意味着无论一个页面被频繁访问还是很少访问，只要它先进入内存，就会一直留在内存中，可能导致内存中驻留的页面并不是最常使用的页面。

（2）FIFO算法可能会导致较高的页面置换开销。当内存块数量有限时，新进入的页面会将最早进入内存的页面置换出去，这可能导致频繁的页面置换操作，增加了磁盘I/O开销。

（3）FIFO算法在增加内存块数量时，缺页率反而可能增加。这是因为FIFO算法只考虑页面进入内存的时间，而不是页面的访问频率，当内存块数量增加时，原本可能被置换出去的页面反而留在内存中，导致更多的页面缺失。

4.1.2. LRU算法优缺点

优点：

（1）LRU算法考虑了页面的访问模式，将最近被访问的页面保留在内存中，从而更有可能满足未来的访问需求。相比于FIFO算法，LRU算法更加关注页面的访问频率，能够更有效地减少页面缺失率。

（2）在一定程度上，LRU算法能够提供较好的性能，尤其是在访问模式具有局部性的情况下。它通常能够较好地利用缓存，减少页面置换的次数，从而提高系统的性能。

缺点：

（1）相比于FIFO算法，LRU算法的实现相对复杂。需要额外开辟空间来记忆页面的访问顺序，以便能够在需要置换页面时快速找到最久未被访问的页面。这增加了算法的实现和管理的复杂性。

（2）为了记录页面的访问顺序，LRU算法需要维护额外的空间，通常是一个较大的缓存

链表或哈希表。这会增加存储开销，并可能占用较多的内存空间。

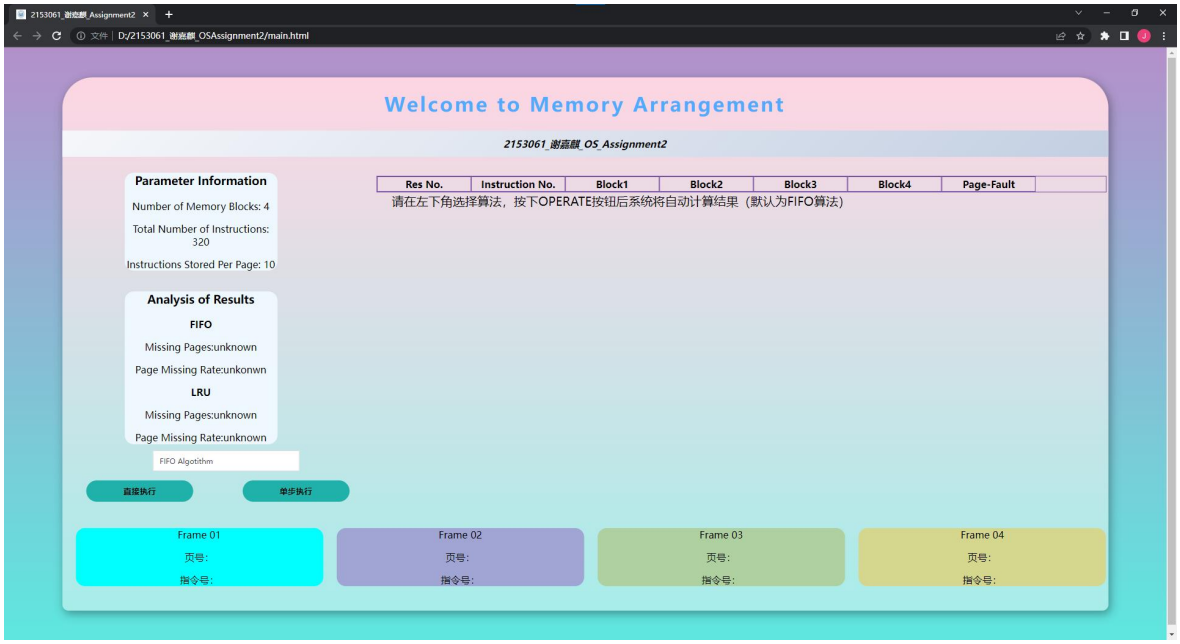
（3）LRU算法需要在每次页面访问时更新页面的访问时间，以保持页面的访问顺序。这会增加一定的时间开销，特别是在访问频繁的情况下，需要频繁地更新访问时间。

4.2. 使用前端三件套的收获

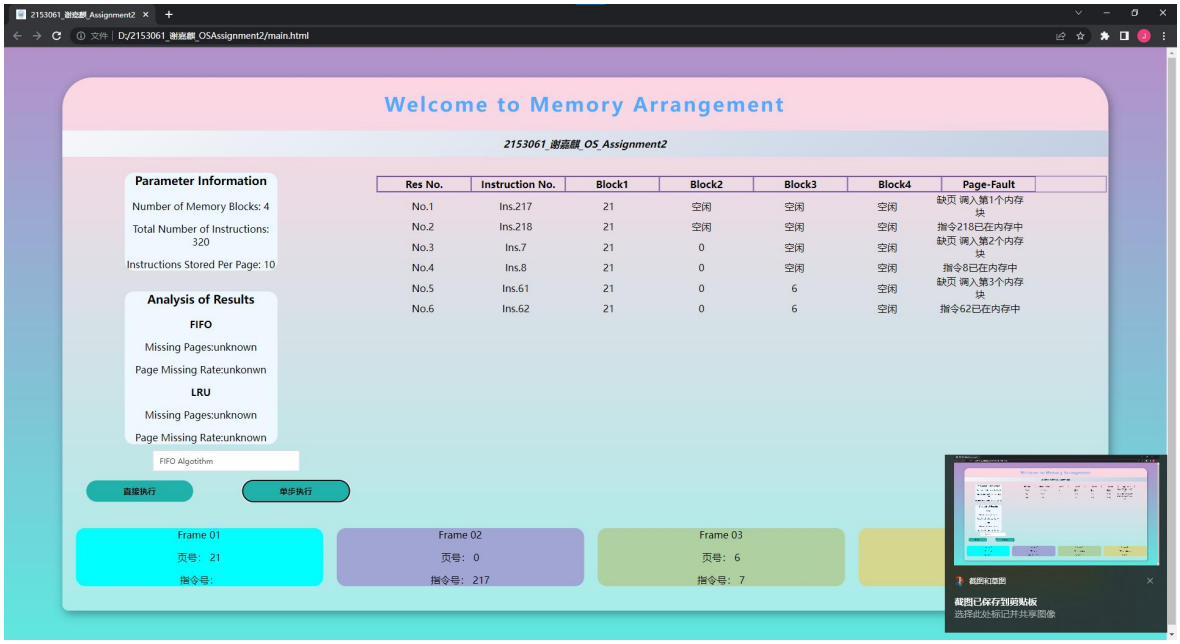
通过本次项目，更进一步熟悉了前端三件套的使用，在布局，交互和动态更新等方面都有了新的收获。同时也发现了一些值得注意的问题，如在HTML内引入jQuery.js需要在引入应用其的.js文件之前；代码顺序执行对函数先后书写顺序由一定要求等问题。

5. 项目展示

5.1. 项目开启界面



5.2. 单步执行FIFO算法



5.3. 直接执行LRU算法