**# 请求调页存储管理方式模拟——说明文档**

**## 项目说明**

**### 项目简介**

通过模拟内存中指令的执行以及缺页置换的过程，本项目通过可视化很好的展现了内存管理的核心思路。

**### 项目功能**

本项目实现了以下的功能：

- **\*\*指令的执行\*\***

  项目模拟了指令顺序执行、随机执行和按照前中后三个区域混合执行的不同策略，比较真实的反映了内存中指令执行的过程

- **\*\*内存页的展示\*\***

  项目绘制了内存中当前页面，从而能够比较清晰的了解内存状况以及当前所正在执行的指令

- **\*\*多种执行方式\*\***

  项目提供了单步执行，以及一步执行到底的连续执行，以便观察指令执行效果

- **\*\*相关信息的展示\*\***

  项目在界面左侧栏展示了当前的缺页数和缺页率，能够比较好的了解该种方式下的执行效率

**### 项目环境**

- **\*\*开发环境\*\***

  Vue + Node.js

- **\*\*运行方法\*\***

  1. 在线运行：https://wangwangwang.website/OS-PageSwapManagement/

  2. 本地运行：

     本项目依赖于 ``` npm ``` 和 ``` Vue CLI ```，运行前请确保环境中已安装[node.js](https://nodejs.org/en/)

     在项目根目录下，运行     npm install

     可以下载项目所依赖的文件包，接下来再通过：

     npm run serve

     可以在本地部署内存管理模拟的网页。完成后依照命令行界面的提示便可在浏览器中测试该模拟器，默认运行在计算机 [8080 端口](http://localhost:8080/)

**## 功能实现**

**### 置换算法**

本项目中实现了两个经典页面置换算法，即**\*\*先进先出(FIFO)\*\***和**\*\*最近最少使用(LRU)\*\***算法。

首先是先进先出算法：

> 先进先出算法是最简单的页面换算法，是指每次有新的[分页](https://baike.baidu.com/item/分页/2888444)需要调入时，会选择调入内存时间最久的分页换出。它简单，容易实现，但这种绝对的公平方式容易导致效率的降低。

本项目中，通过使用一个指针指向最早进入内存的页面。当需要进行置换时，取出该指针所指向的页即可。代码如下：

```javascript

FIFO(){

  let outPage=this.FIFOPointer;//指向最早进入内存页面

  ++this.FIFOPointer;

  if(this.FIFOPointer==this.innerPage.length){

    this.FIFOPointer=0;

  }

  return outPage;

}

```

而LRU算法的原理则如下所示：

> 最近最少使用算法，是一种常用的页面置换算法，选择最近最久未使用的页面予以淘汰。该算法赋予每个[页面](https://baike.baidu.com/item/页面/5544813)一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t，当须淘汰一个页面时，选择现有页面中其 t 值最大的，即最近最少使用的页面予以淘汰。

本项目中，通过数组LRUTable记录了每一个页面最近被使用的时刻。当需要进行置换时，查询该表并取出最早被使用的一个页面即可。代码如下：

```javascript

LRU(){

  let minData=this.sumInstruction+1;

  let minIndex=-1;

  for(let i=0;i<this.LRUTable.length;++i){

    if(this.LRUTable[i]<minData){

      minData=this.LRUTable[i];

      minIndex=i;

    }

  }

  return minIndex;

```

两种算法执行320条指令后，缺页率分别为64%和58%。

这一结果也符合我们的预期，虽然本项目中指令模拟和现实中的指令不尽相同，但也一定程度上反映出LRU置换算法比FIFO置换算法有着更好的表现。

**### 指令序列的模拟**

为了模拟出：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。

本项目通过先在0~319中产生一个随机数i，作为第一条指令的序号，之后顺序执行下一条指令i+1；

之后在0~i-1中产生一个随机数作为指令序号j，即在跳转到前地址部分；

之后在i+2~319中产生一个随机数执行，代表跳转到后地址，再顺序执行下一条指令。

**### 其他**

本项目中，定义了以下函数：

| 函数           | 说明                     |

| -------------- | ------------------------ |

| restart()      | 重置指令执行状态         |

| findPos()      | 寻找下一条需要执行的指令 |

| nextInstruct() | 单步执行下一条指令       |

| keepInstruct() | 连续执行指令             |

| FIFO()         | 先进先出置换算法         |

| LRU()          | 最近未使用置换算法       |

| instructInfo() | 查看项目说明             |

![2NJeaQ.jpg](https://z3.ax1x.com/2021/06/05/2NJeaQ.jpg)

指令执行完毕时，弹窗提醒：

![2NJKGn.jpg](https://z3.ax1x.com/2021/06/05/2NJKGn.jpg)

**## 项目总结**

**### 项目亮点**

- 实现了两种置换算法和不同指令执行次序的模拟

- 界面美观，能够比较好的展示内存模拟过程

- 部署在网页，方便学习交流

- 指令执行过程中，会亮起当前正在执行的指令，便于观察

- 提示信息清晰

**### 项目改进方向**

LRU是实际应用比较广泛的一种置换算法，在本项目中也有着比较好的表现性能。但是在未来可以考虑其他置换算法的应用，从而进一步降低缺页率。

此外，本项目中界面仅针对于网页端，尚未对手机浏览器做出相应适配。未来开发，可以考虑向这一方面改进。