**内存管理-说明文档**



项目实现了以下功能：

* **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。

* **内存页的显示**

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**

项目中可以选择单步执行指令和连续执行指令，同时还可以选择重置执行结果，方便进行多次试验以及观测试验结果。

* **多种分配算法和指令执行顺序的选择**

项目内置了FIFO和LRU两种替换算法以及3中指令执行顺序可供选择，方便对比。

* **执行过程的记录**

项目的界面中有一个展示所有指令的执行顺序以及页面替换状况的输出框，便于程序的调试以及程序完成后对程序正确性的校准。

* **执行结果的展示**

项目在界面左下侧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**

C#.NET Wondows窗体

Win10

Visual Studio 2019

* **运行环境**

目前只在安装了Visual Studio 2019 .NET开发环境的Win10环境中进行了测试，为保证稳定运行请保持环境一致。

* **运行方法**

双击PageSwapper.exe即可运行本项目。

**功能实现**

**界面介绍**

内存中存放的页号

记录指令执行信息



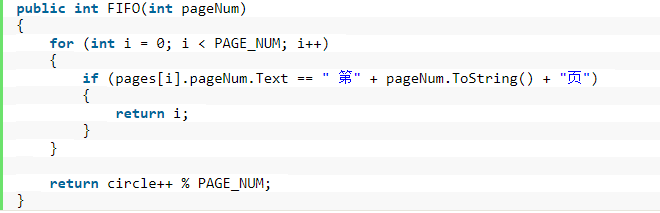
内存中的页面，红色表示被占用

**替换算法**

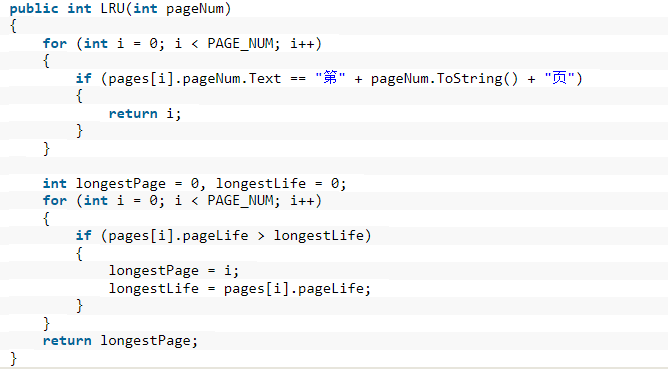
本项目中实现了两个页面替换算法，包括先进先出算法(FIFO)和最近最少使用算法(LRU)。

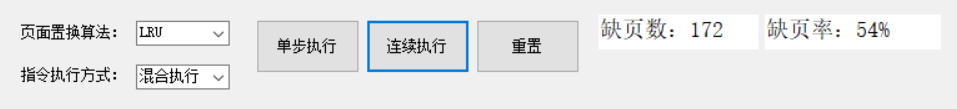
首先是FIFO页面置换算法, 也就是先进先出的意思。这和我们现实生活中的排队方式很相似, 先进队伍的人会先买到票, 然后先从队伍中离开。如果使用FIFO算法作为页面置换算法, 缓存空间大小是三个页面时, 一次进入Page1, Page2, Page3。当Page4要进入缓存时, 操作系统将会把Page1清除出缓存, 将Page4加入至缓存中。如果再有Page5要进入缓存时, 操作系统会将Page2清除出缓存空间, 以此类推。

本项目中用一个四维数组和一个计数器简单的实现了FIFO算法：



第二种是LRU算法，这种算法的基本思想是：利用局部性原理，根据一个作业在执行过程中过去的页面访问历史来推测未来的行为。它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问。所以，这种算法的实质是：当需要淘汰一个页面时，总是选择在最近一段时间内最久不用的页面予以淘汰。



两种算法执行320条指令后，缺页率分别为59%和54%。虽然两者的差距并不大，但是LRU在多数情况下还是比FIFO的效率要高。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。

**指令执行顺序模拟**

* 随机执行：随机选取一条指令执行 。
* 顺序执行：从0-319顺序执行所有指令。
* 混合执行：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。

**项目亮点**

* **实现了两种替换算法和不同的指令顺序的模拟。**
* **界面简洁美观，可以直观地观测到内存模拟过程。**
* **指令执行过程中，会亮起当前正在执行的指令，便于观察**
* **运用了面向对象的方法，代码复用率高。**

**项目改进方向**

* **可以尝试加表现更好的替换算法**
* **由于框架的使用，仅限于在Windows平台运行。在未来项目开发过程中，可以考虑采取其他开发工具，使得项目可移植性更好**