**操作系统——内存管理项目2**

**请求分页分配方式**

**学号：2151298**

**姓名：杨滕超**

**班号：42036902**

**指导老师：张惠娟**

目录

**[操作系统——内存管理项目2 1](#_Toc5737)**

**[1. 项目介绍 2](#_Toc19532)**

[1.1. 项目背景需求 2](#_Toc8365)

[1.2. 项目目的 2](#_Toc22300)

**[2. 需求分析 2](#_Toc6151)**

**[3. 算法分析 2](#_Toc16686)**

[3.1. FIFO算法 2](#_Toc17560)

[3.2. LRU算法 3](#_Toc25865)

**[4. 实现功能 3](#_Toc16686)**

[4.1. 指令数量、内存页面数量以及每页指令数量设置 3](#_Toc17560)

[4.2. 指令单步执行 3](#_Toc17560)

[4.3. 指令连续执行开始与暂停 3](#_Toc17560)

[4.4. 重置 3](#_Toc17560)

[4.5. 相关信息显示 3](#_Toc17560)

**[5. 开发环境 4](#_Toc19620)**

**[6. 界面设计 4](#_Toc621)**

[6.1. 内存页面区域 4](#_Toc31378)

[6.2. 指令执行信息显示区域 4](#_Toc16470)

[6.3. 操作区域 5](#_Toc13409)

**[7. 系统设计 5](#_Toc14089)**

[7.1. 逻辑设计 5](#_Toc11461)

[7.2. 全局变量设置 6](#_Toc11461)

[7.3. 类设计 7](#_Toc11461)

**[8. 算法 8](#_Toc32311)**

[8.1. 随机下一条指令编号 8](#_Toc22940)

[8.2. 页面交换 9](#_Toc19471)

[8.3. 执行下一条指令 1](#_Toc19471)0

**[9. 程序运行 1](#_Toc30342)0**

[9.1. 单步执行/连续执行 10](#_Toc22940)

[9.2. 执行完成 1](#_Toc22940)1

[9.3. 连续执行过程中点击其他按钮 1](#_Toc22940)1

1. **项目介绍**
   1. **项目背景需求**

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。

置换算法可以选用FIFO或者LRU算法。

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. **项目目的**

通过模拟内存页面的调度，学习并加深对操作系统中调页过程的理解，以及调页策略的选择优劣比较。

1. **需求分析**

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。

置换算法可以选用FIFO或者LRU算法。

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. **算法分析**

在进程运行的过程中，随着编程要求的不断发展以及数据处理愈加复杂，往往会出现大程序、小内存的问题。此时就需要对进程进行划分，使得进程部分地进入内存，从而使得内存扩充，大程序得以在小内存中运行。但在运行的过程中，难免遇到这样一个问题：所要访问的页面不在内存，产生缺页中断，此时需要把他们从外存中调入内存。若此时操作系统发现内存中没有空闲页面，则操作系统必须选择内存中的一个页面移出，以便于为将要调入的页面提供空间。此过程则需要页面置换算法。其中包括：先进先出置换算法（FIFO）、最佳置换算法（OPT）、最近最久未使用算法（LRU）、最不常用置换算法（LFU）以及轮询算法（CLOCK）。下面将介绍本项目使用的两种置换算法。

**3.1. FIFO算法**

FIFO算法，即先进先出置换算法，是一种较为简单的置换算法，也是一种提出较早的算法。其核心思路为：操作系统总是会选择在主存中停留时间最长的一个页面进行置换，即最先进入内存的页。其中算法设计的理由依据为：认为最早调入内存的页，其不在被使用的可能性比刚调入内存的可能性大。实施方法为：建立一个先进先出的队列，一个页面每调入内存，就进入队列。当需要进行页面置换的时候，弹出队首的那个页面，即将队首的页面进行置换。

然后这种算法只是对于按照线性顺序访问地址空间时才是最理想的，否则效率并不高。因为根据局部性原理，那些经常被访问的页，往往在主存中页停留最久。因此在实际应用的时候，常常会出现抖动现象，即对于刚刚被替换出去的页，立即又要被访问，需要将它调入，因无内存空间又要替换另一页，而后者又是即将被访问的页，于是造成了操作系统需要花费大量的时间忙于进行这种频繁的页面交换，导致系统的实际效率十分低下，严重导致系统瘫痪。除此之外，FIFO算法还会产生Belady现象，即如果对一个进程未分配它所要求的全部页面，有时候会出现分配的页面数量增多但缺页率反而会提高的异常现象。

**3.2. LRU算法**

对于理想情况下使用的OPT算法，LRU退一步——利用最近不久的数据预测未来。根据局部性原理，以最近的过去作为对不久将来的近似，那么可以把过去最长一段时间里不被使用的页面置换掉。其中需要记录的数据为内存中每个页面最后使用的时间。

相对于FIFO算法，LRU算法避免了长时间驻留在缓存中但却很少被访问的数据，从而降低了缓存操作的开销。从而具有更高的性能，更好的稳定性。

1. **实现功能**

**4.1. 指令数量、内存页面数量以及每页指令数量设置**

程序一开始可以通过键盘输入指令数量、内存页面数量以及每页指令数量这些参数，并且程序会进行初步检查所输入的数据是否合理，确认合理后才进一步进入主界面。当然直接回车使用默认参数的设置，即项目ppt中所要求的参数数据。

**4.2. 指令单步执行**

通过点击单步执行按钮，可以单步执行一条指令。若执行到最后一条指令之后仍然点击，则弹出消息窗口提示进行重置。

**4.3. 指令连续执行开始与暂停**

通过点击开始/暂停连续执行按钮，可以开始或结束连续执行指令，同时可以通过滑动条设置每条指令执行的时间间隔。若执行到最后一条指令之后仍然点击，则弹出消息窗口提示进行重置。

**4.4. 重置**

通过点击重置按钮，可以重置当前指令执行的进度，从而重新开始执行。

**4.5. 相关信息显示**

每一条指令指令执行过后，指令执行的相关信息会输出到程序界面的对应区域中。其中信息包括：当前执行步数、指令编号、是否缺页、换入页号以及换出页号。同时在侧边信息栏中实时显示当前执行的指令编号、作业指令总数、每页存放指令数、内存页数、页面置换算法的选择、执行进度、缺页数、当前执行步数、缺页率、指令执行间隔调节等信息。

1. **开发环境**

开发环境：Windows\_NT x64 10.0.19044

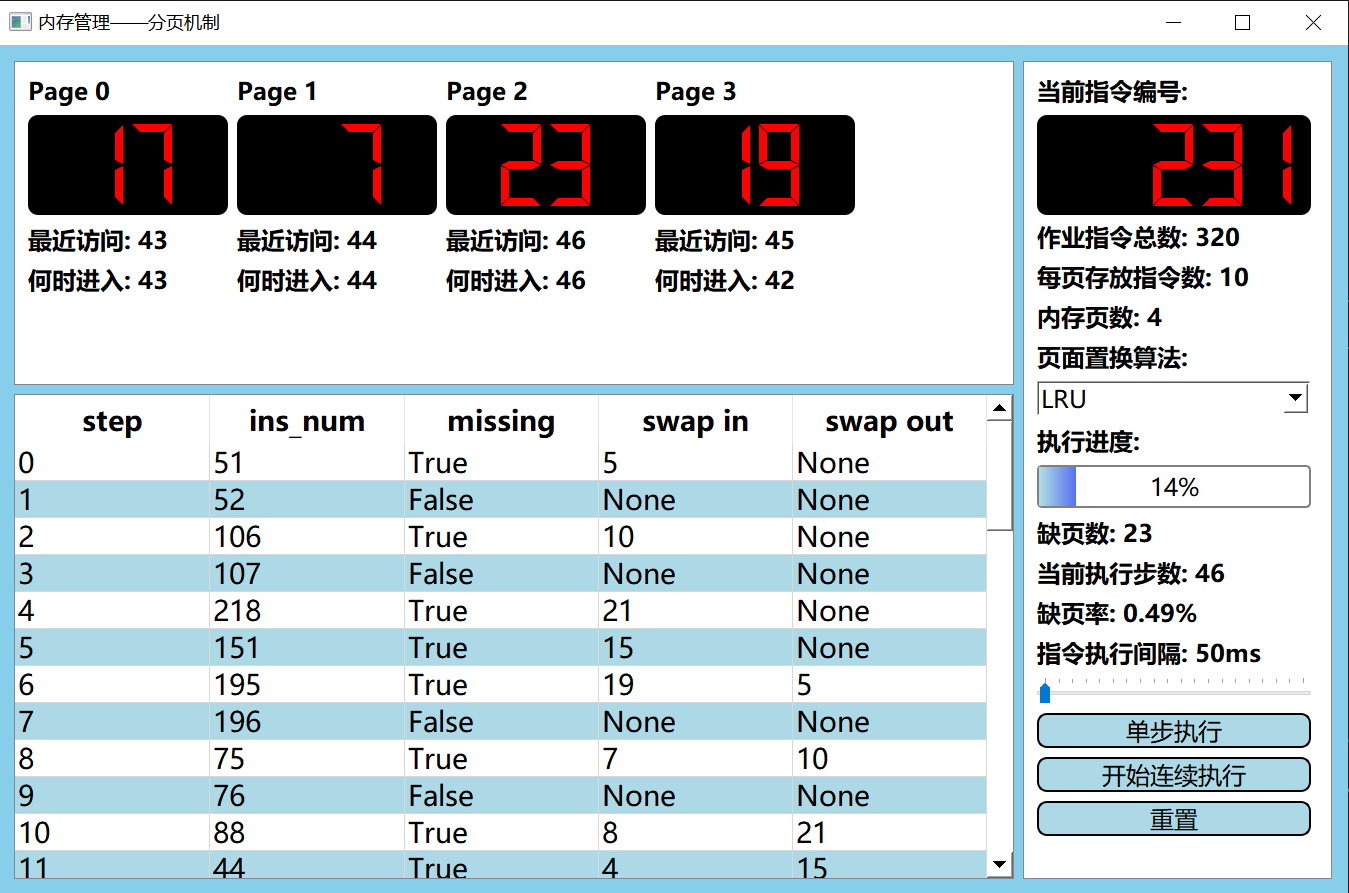
开发软件：VSCode1.76.0

开发语言：Python3

IDE：Python 3.10 64-bit

主要引用模块：PyQt5

1. **界面设计**



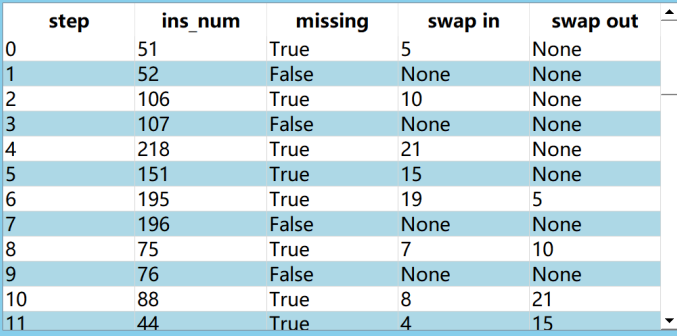
本项目设置为界面设置3个区域，分别表示不同功能区。下面将按照从上到下、从左到右的顺序介绍。

**6.1. 内存页面区域**



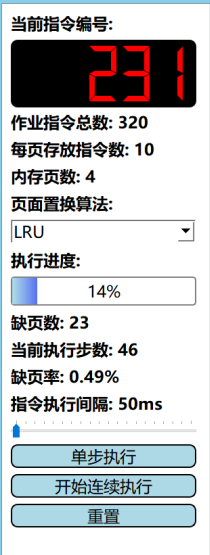
本区域提供内存页号以及最近使用以及何时进入的执行步数的显示，实时显示内存中页面的物理页号。

**6.2. 执行指令信息显示区域**



本区域提供每一步对应的指令编号、是否缺页、换入页号以及换出页号。动态实时地显示每一步的相关信息。

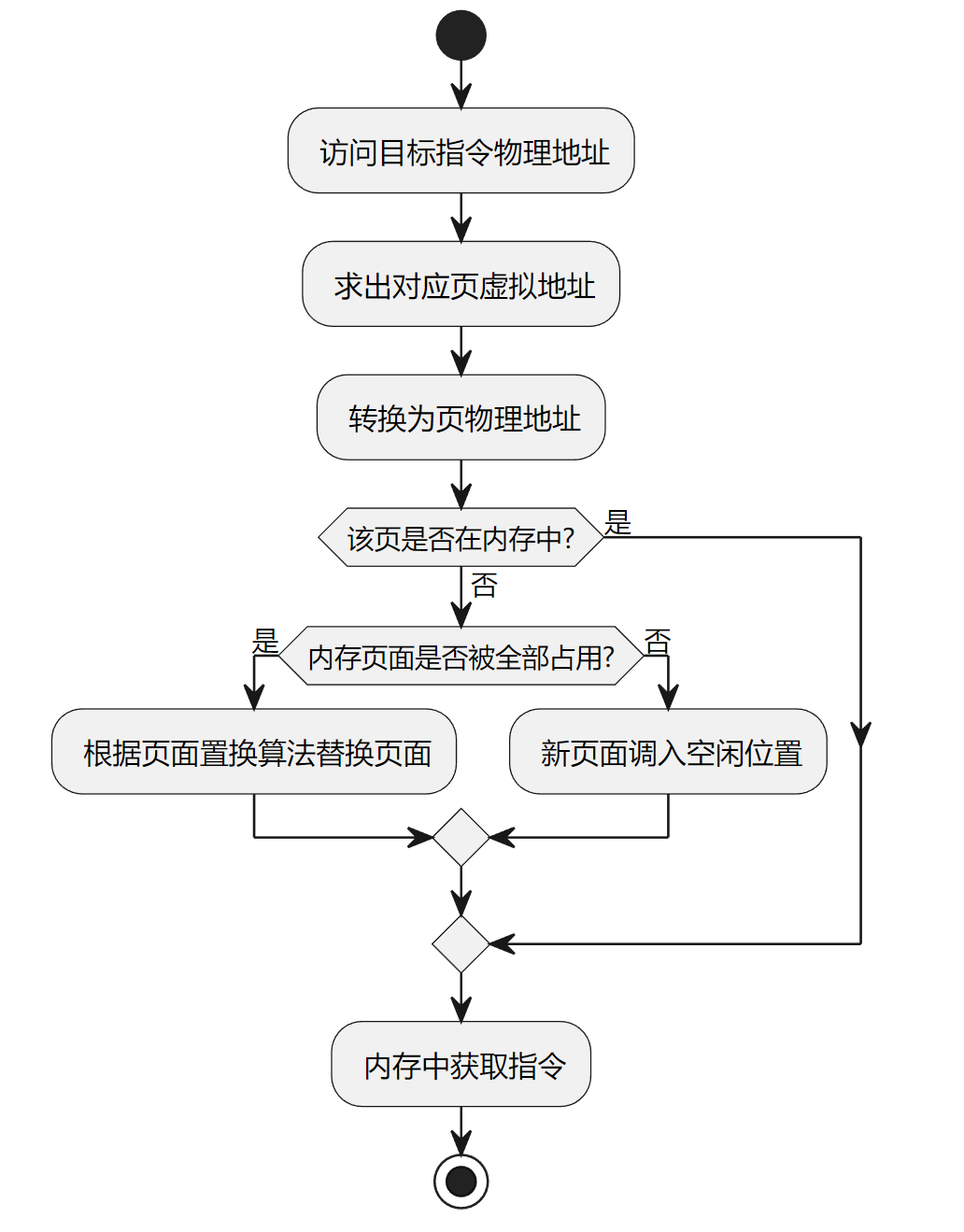
**6.3. 操作区域**

这里实时显示当前执行的指令编号，其下为所设置的参数信息。接着是可以选择使用的置换算法的下拉框，以及作业执行的进度。然后是对于缺页数的记录以及相关信息。最后是执行按钮的设置，分别是单步执行、连续执行、重置操作。

1. **系统设计**

**7.1. 逻辑设计**

指令执行过程如下图所示：



**7.2. 全局变量设置**

|  |  |
| --- | --- |
| **变量名** | **含义** |
| **INSTRUCTION\_NUM** | **程序将要执行的指令总数** |
| **INSTRUCTION\_PER\_PAGE** | **单个内存页面上的指令数量** |
| **PAGE\_NUM** | **内存中可用总页面数** |
| **MAX\_INTERVAL** | **指令之间最长时间间隔** |
| **GOON** | **控制程序是否应该继续执行** |
| **WIDTH1** | **左边区域宽度** |
| **missing\_page\_num** | **缺页次数** |
| **cur\_step** | **当前步数记录** |
| **previous\_ins\_id** | **上一条指令编号** |
| **begin\_loop** | **指令是否连续执行** |
| **interval** | **指令之间时间间隔** |
| **selected** | **所选择的算法** |
| **finished** | **程序是否完成** |
| **output** | **指令执行信息** |
| **pages** | **内存中页面信息** |
| **PAGE\_NUM\_PER\_ROW** | **显示每行页数** |
| **WIDTH, HEIGHT** | **界面尺寸** |
| **INIT\_NUM** | **初始化数字** |

**7.3. 类设计**

**7.3.1. Page类**

其中包括数据成员记录虚拟页号以及进入的步数、最近使用的步数。

# 记录页相关信息

class Page:

    def \_\_init\_\_(self, pid = -1, es = None, rs = None):

        self.page\_id = pid

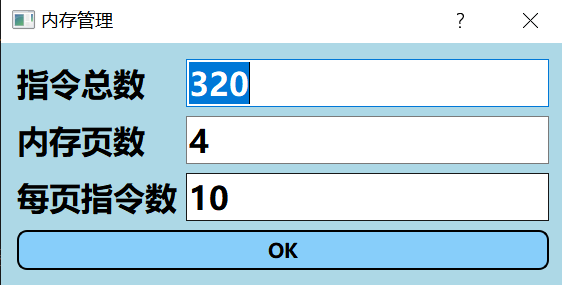
        self.enter\_step = es

        self.recent\_used\_step = rs

**7.3.2. myDialog类**

该类继承于QDialog类，在程序运行一开始显示，并接受合理数字输入，由此决定任务指令数量、内存页数以及每页容纳多少指令数量。初始值分别为320、4和10。若直接关闭该窗口程序会直接结束；若点击OK按钮或回车，系统会检查输入是否合理，若合理则直接进入主程序，否则继续等待正确输入。

运行如下图所示：



**7.3.3. myWin类**

该类继承于QWidget类，主要用于主界面的显示，由此将Page类联系起来。

包含内部方法如下：

|  |  |
| --- | --- |
| **方法名** | **作用** |
| setup\_UI() | 界面绘制 |
| reset\_clicked() | 重置按钮按下处理函数 |
| single\_step\_clicked() | 单步执行按钮按下处理函数 |
| con\_step\_clicked() | 连续执行按钮按下处理函数 |
| delay() | 延迟函数 |
| change\_interval(val) | 指令执行时间间隔改变函数 |
| selected\_change(algorithm\_text) | 算法选择下拉框改变函数 |
| table\_clear() | 清除表格输出信息函数 |
| finish\_ins() | 判断是否完成作业函数 |
| con\_other\_btn() | 连续执行中其他按钮按下函数 |
| update() | 更新UI界面函数 |

1. **算法**

**8.1. 随机下一条指令编号**

# 随机生成下一条指令的编号

def get\_next\_ins():

    global previous\_ins\_id

    res = -1

    # 不是第一条

    if previous\_ins\_id is not None:

        random\_num = random()

        if random\_num < 0.5:

            if previous\_ins\_id + 1 < INSTRUCTION\_NUM:

                ans = previous\_ins\_id + 1

            else:

                ans = floor(random() \* INSTRUCTION\_NUM)

        elif 0.5 <= random\_num <= 0.75:

            ans = floor(random() \* previous\_ins\_id)

        else:

            ans = floor(random() \* (INSTRUCTION\_NUM - previous\_ins\_id)) + previous\_ins\_id

    else:

        ans = floor(random() \* INSTRUCTION\_NUM)

    # 更新上一条指令记录

    previous\_ins\_id = ans

    output['ins\_num'] = ans

    print('next idx is %d' % ans)

    return ans

算法思路

首先判断是否是第一条指令，若是则随机生成指令总数中的任意一条。若不是，则需要根据上一条指令进行判断。其中50%的概率顺序执行；25%的概率随机生成当前指令的前面指令，再使用随机数随机在前面的指令中选取一条；25%的概率生成当前指令的后面指令，方法同上。

**7.2. 页面交换**

# 将页交换

def swap(page\_id, algorithm=selected):

    global output

    print('将page: %d 换入' % page\_id)

    # 找到最先进来的或最近未使用的

    min\_step = INSTRUCTION\_NUM

    target\_page = None

    for page in pages:

        if page.enter\_step < min\_step:

            min\_step = page.enter\_step if algorithm == 'FIFO' else page.recent\_used\_step

            target\_page = page

    pages[pages.index(target\_page)] = Page(page\_id, cur\_step, cur\_step)

    output['swap in'] = page\_id

    output['swap out'] = target\_page.page\_id

    print('将page: %d 换出' % target\_page.page\_id)

根据所选择的算法，选择在内存中的页面中所进入内存的步数或最近使用的步数中最小的，即为最先进入内存的页或最久未被使用的页作为FIFO或LRU算法选择置换出内存的页面，然后将外存中要调入的页面换入。

**7.3. 执行下一条指令**

def jump\_next(next\_ins\_id):

    global cur\_step, missing\_page\_num, output

    output['step'] = cur\_step

    found = False

    page\_id = floor(next\_ins\_id / INSTRUCTION\_PER\_PAGE)

    # 检查是否已经在内存中的页表

    for page in pages:

        if page.page\_id == page\_id:

            # 找到了更新最近使用时间

            page.recent\_used\_step = cur\_step

            found = True

            break

    if found:

        output['missing'] = False

        print('内存中找到 %d, 不用换' % page\_id)

    else:

        output['missing'] = True

        missing\_page\_num += 1

        # 检查内存中是否还可以调入页面

        get\_in = False

        for idx, page in enumerate(pages):

            if page.page\_id == -1:

                print('有空位')

                pages[idx] = Page(page\_id, cur\_step, cur\_step)

                output['swap in'] = page\_id

                get\_in = True

                break

        # 没有空位，需要swap

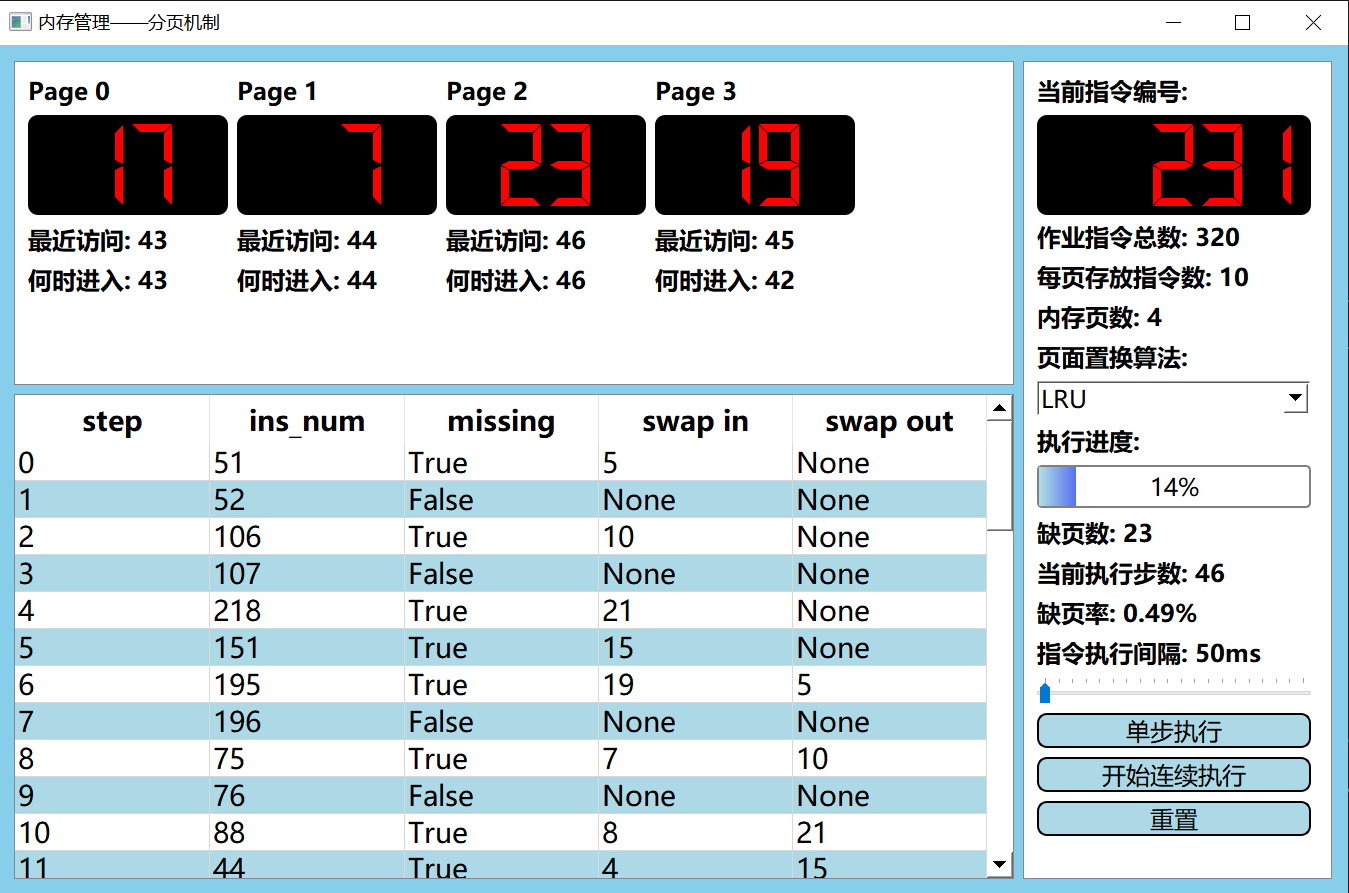
        if not get\_in:

            swap(page\_id)

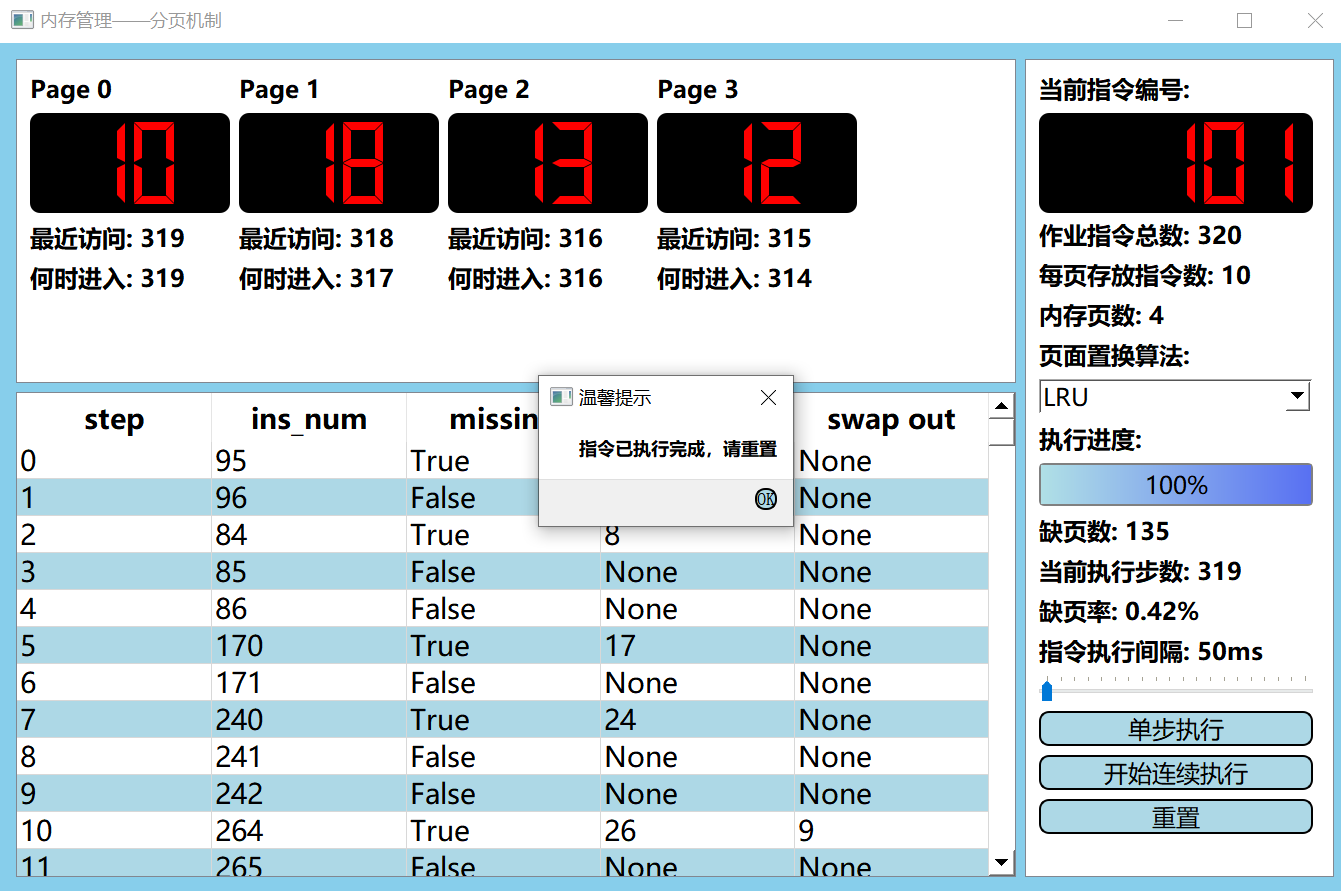
首先计算下一条指令所对应的虚拟页号，然后检查内存中的是否已经装入该页，若已经装入则无需换入。若没有，进一步检查内存中是否还有空闲空间，若有则直接装入；若没有需要调用页面置换算法swap函数进行页面的置换。

**8. 程序运行**

**8.1. 单步执行/连续执行**



**8.2. 执行完成**



**8.3. 连续执行过程中点击其他按钮**

