**内存管理模拟系统设计报告**

**基本任务**

1. **动态分区分配方式**

数据结构、分配算法

加深对动态分区存储管理方式及其实现过程的理解。

假设初始态下，可用内存空间为640K，并有下列请求序列，请分别用**首次适应算法**和**最佳适应算法**进行内存块的分配和回收，并显示出每次分配和回收后的空闲分区链的情况来。

作业1申请130K

作业2申请60K

作业3申请100k

作业2释放60K

作业4申请200K

作业3释放100K

作业1释放130K

作业5申请140K

作业6申请60K

作业7申请50K

作业6释放60K

1. **请求分区分配方式**

页面、页表、地址转换

页面置换过程

加深对请求调页系统的原理和实现过程的理解。

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

**界面设计**

模拟系统界面使用python自带的PyQt5图形界面库，用户可以通过最上方不同的栏在动态分区分配和页面置换之间切换。

1. **动态分区分配界面**

用户可以在“分配算法”栏里切换不同算法，在“操作序列”框内进行作业分配释放操作，下方显示内存状态，包括起始地址、大小、状态等。“操作日志”栏显示作业分配或释放操作是否成功，最下方可以进行重置。

1. **页面置换界面**

用户可以在“分配算法”栏里切换不同算法，在“控制”框内进行操作，首先进行指令序列生成，生成指令及其访问地址并显示指令序列。可以选择“单步执行”或是“自动运行”进行页面置换。“执行状态”栏显示指令、访问地址 、页号 、偏移 、是否命中或是缺页以及物理地址等。“结果”栏显示使用的算法类型，总指令数，缺页次数以及缺页率。

**实现方式**

**1. 技术栈**

* **语言**：Python 3.8.1
* **图形界面库**：PyQt5
* **实现算法**：动态分区：首次适应算法和最佳适应算法

请求分区：FIFO算法和LRU算法

**2. 动态分区分配模块：首次适应算法和最佳适应算法**

* **首次适应算法(First Fit)**：遍历内存分区，找到第一个大小符合请求的分区进行分配，如果大于需求则分割成两块。

    def allocate\_first\_fit(self, job\_id, size):

        """首次适应算法"""

        for i, block in enumerate(self.memory\_blocks):

            if block.is\_free and block.size >= size:

*# 找到足够大的空闲分区*

                if block.size == size:

*# 如果分区大小正好等于请求大小，直接分配*

                    block.is\_free = False

                    block.job\_id = job\_id

                else:

*# 分割空闲分区*

                    self.memory\_blocks.insert(i + 1, MemoryBlock(block.start + size, block.size - size))

                    block.size = size

                    block.is\_free = False

                    block.job\_id = job\_id

                return True

        return False  *# 没有找到合适的空闲分区*

* **最佳适应算法(Best Fit)：**遍历所有空闲分区，选择剩余空闲最小的那一块分配。

 def allocate\_best\_fit(self, job\_id, size):

        best\_fit\_index = -1

        min\_fragment = float('inf')

        for i, block in enumerate(self.memory\_blocks):

            if block.is\_free and block.size >= size:

                fragment = block.size - size

                if fragment < min\_fragment:

                    min\_fragment = fragment

                    best\_fit\_index = i

        if best\_fit\_index != -1:

            block = self.memory\_blocks[best\_fit\_index]

            if block.size == size:

*# 如果分区大小正好等于请求大小，直接分配*

                block.is\_free = False

                block.job\_id = job\_id

            else:

*# 分割空闲分区*

                self.memory\_blocks.insert(best\_fit\_index + 1, MemoryBlock(block.start + size, block.size - size))

                block.size = size

                block.is\_free = False

                block.job\_id = job\_id

            return True

        return False  *# 没有找到合适的空闲分区*

**3. 请求分页模块：FIFO算法和LRU算法**

* **FIFO 算法**：替换 load\_time 最早的页

 def \_replace\_page\_fifo(self):

        """FIFO页面置换算法"""

*# 寻找最早加载的页面*

        min\_load\_time = float('inf')

        min\_frame = -1

        for i, page\_number in enumerate(self.frames):

            if self.pages[page\_number].load\_time < min\_load\_time:

                min\_load\_time = self.pages[page\_number].load\_time

                min\_frame = i

        return min\_frame, self.frames[min\_frame]

* **LRU 算法：**替换 last\_access\_time 最早的页

 def \_replace\_page\_lru(self):

        """LRU页面置换算法"""

*# 寻找最久未使用的页面*

        min\_access\_time = float('inf')

        min\_frame = -1

        for i, page\_number in enumerate(self.frames):

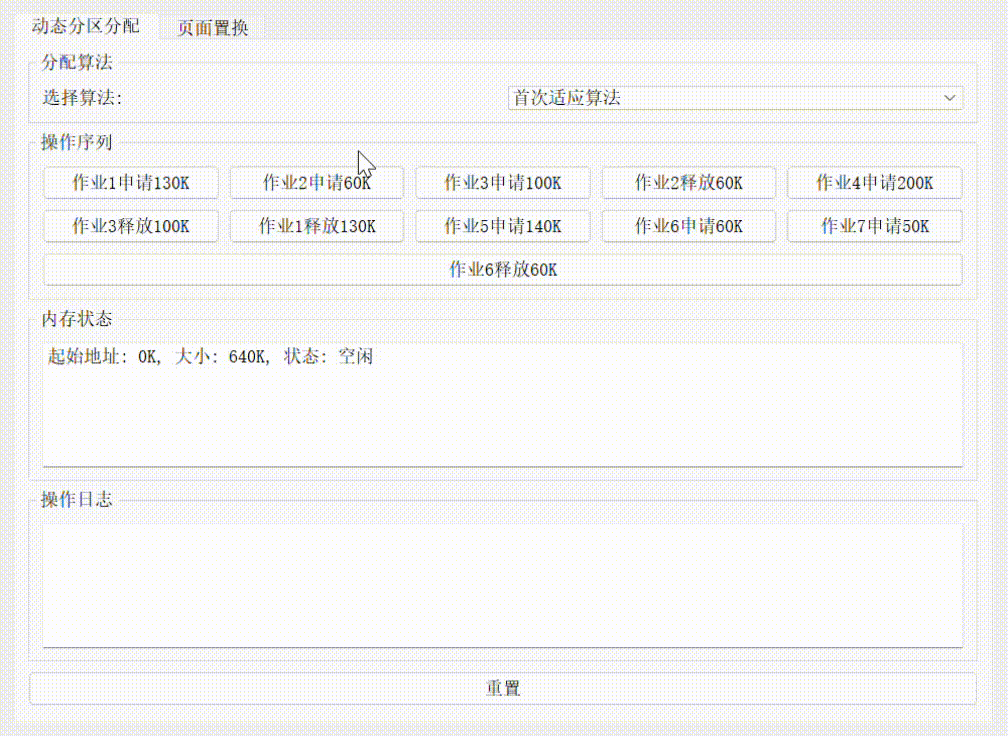
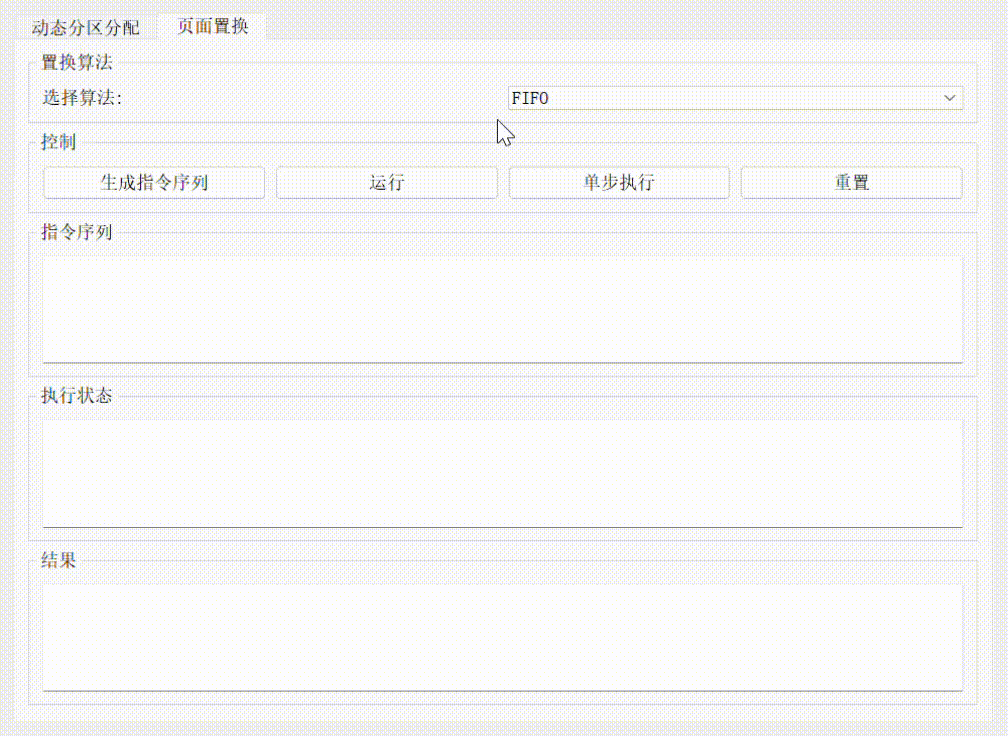
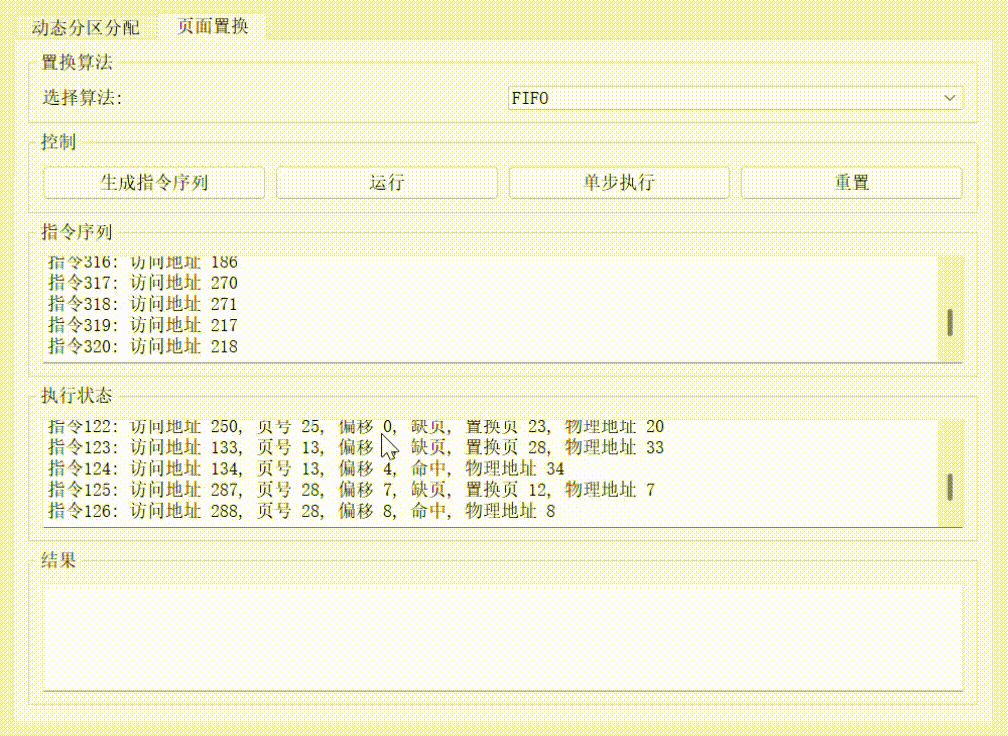
            if self.pages[page\_number].last\_access\_time < min\_access\_time:

                min\_access\_time = self.pages[page\_number].last\_access\_time

                min\_frame = i

        return min\_frame, self.frames[min\_frame]

**附件**

1. **动态分区分配演示**
2. **请求分区分配演示**