# 内存管理项目

# 请求调页存储管理方式模拟

姓名：李若冰

学号：2251657

指导教师：张慧娟

学院/专业：软件学院软件工程专业

日期：24/5/22

目录

**[内存管理项目 1](#_Toc28413)**

**[请求调页存储管理方式模拟 1](#_Toc24685)**

**[1项目需求 2](#_Toc22724)**

[1.1 基本任务 2](#_Toc10037)

[1.2 模拟过程 2](#_Toc32049)

[1.3开发环境 3](#_Toc16453)

**[2项目展示 3](#_Toc19745)**

[2.1项目界面简介： 3](#_Toc29874)

[2.2界面展示 4](#_Toc25154)

**[3.设计方案 6](#_Toc16566)**

[３.１被模拟执行的作业(Task 类) 6](#_Toc29730)

[３.２为作业分配页面的管理器(MyManager 类) 6](#_Toc24827)

[３.３可视化界面(MainWindow 类) 9](#_Toc27333)

[３.４负责调用管理器、更新界面、响应用户输入的程序主逻辑部分(MyThread 类)。 9](#_Toc27475)

**[４. 文件架构以及运行 12](#_Toc8983)**

# 1项目需求

## 1.1 基本任务

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。

## 1.2 模拟过程

* 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。
* 置换算法选用 FIFO 或者 LRU 算法
* 作业中指令访问次序按照下面原则形成：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。

## 1.3开发环境

开发环境:Windows 11

开发软件:pycharm

开发语言:python 3.8

开发框架:pyqt5

# 2项目展示

## 2.1项目界面简介：

1. 在开始执行一次模拟之前，用户可以设定采用的页面置换算法(默认 FIFO)。置换算法包括 FIFO 与 LRU。
2. 设定仅在一次模拟开始前生效，模拟过程中更改设定并不会生效。
3. 当用户点击开始后，一次模拟即开始，直到执行完设定的指令总数或者用户按下重置按钮后，一次模拟结束。
4. 界面的左边分别展示分配给作业的四个页，每个页存储 10 条指令，竖向的十个按钮为一组，表示内存中的一页。每个按钮上显示该按钮存放的指令的地址(即 0 -319)，每组按钮的上方显示该内存页存放的指令是作业的哪一页。四组按钮的右侧显示当前发生缺页情况的次数与当前的缺页率。
5. 每次执行指令后，以红色高亮按钮，显示本次执行的指令在内存中的位置。
6. 当指令正在执行，右下方会弹出“指令正在执行！！！”。当全部指令执行完成后下方会弹出“全部指令已完成!!!”的提示。并且会清空左侧的内容，右侧会显示缺页率和缺页数以供参考。

## 2.2界面展示

开始界面：



正在执行的界面：





执行结束的界面：





重置界面：



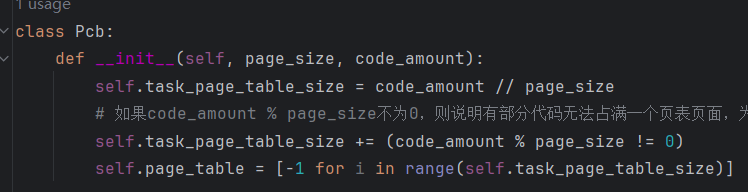
# 3.设计方案

## ３.１被模拟执行的作业(Task 类)

主要是模拟一份作业的指令执行过程。Task 类中包含一个页表，用于记录模拟作业的每一页在模拟分配的内存中的页面位置，未被分配内存则记为-1。

Manager 每一次调用 Task 的执行函数，Task 都会向 Manager 返回当前要执行的模拟作业的指令的地址(即 0- 319)、该代码在模拟内存中的页面位置、该代码位于模拟作业的第几个页面，并按照项目要求的规则在 Task 中生成下一行指令的地址。

* PCB主要任务是分配页表，从而从而在可视化上面能够正确显示：



* Task主要任务是完成地址模拟调用逻辑：



## ３.２为作业分配页面的管理器(MyManager 类)

实现了一个简单的内存管理器 MyManager。它支持两种不同的内存页面管理算法:

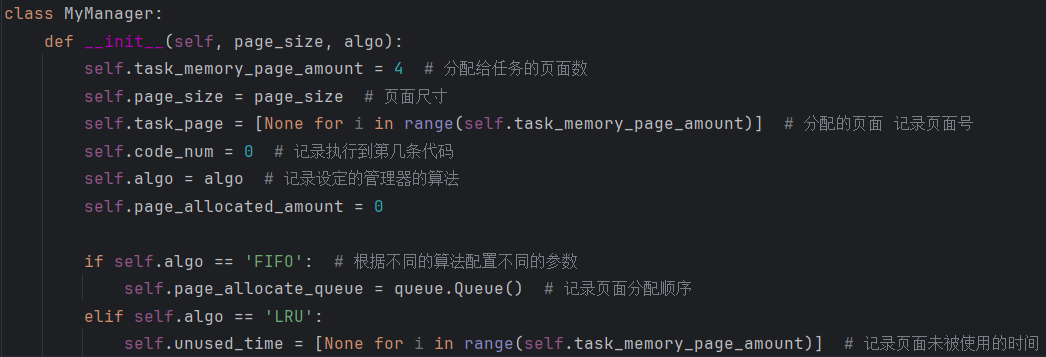
* **FIFO (First-In-First-Out)**: 页面调换时选择最早分配的页面进行调换。使用一个 queue.Queue 来记录分配顺序。
* **LRU (Least Recently Used)**: 页面调换时选择最久未使用的页面进行调换。使用一个列表来记录每个页面的"未被使用时间"。

主要功能包括:

* pageSwap: 实现页面调换的功能。将旧页面从内存中移除,并将新页面加入内存。更新任务的页表。
* allocateEmptyPage: 在内存中分配一个空闲的页面,并更新任务的页表。
* add\_unused\_time: 用于 LRU 算法,更新每个页面的"未被使用时间"。
* runTask: 根据选择的算法 (FIFO 或 LRU) 执行任务。它会判断当前要执行的代码是否在内存中,如果不在则进行页面分配或调换。同时记录一些可视化界面所需的信息。

这个内存管理器提供了两种常见的内存页面管理算法,可以在执行任务时根据需要进行选择。它能够管理任务的内存页面,并提供相关的日志信息供可视化界面使用。

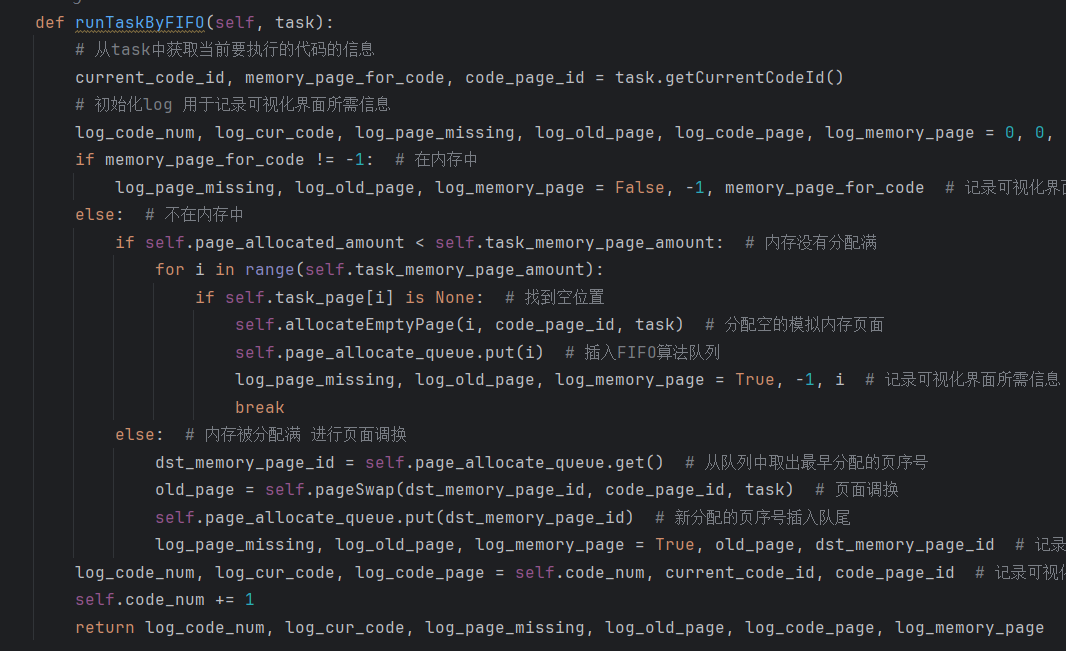
类的初始化：



实现两个算法基础类的需要



FIFO算法实现：



LRU算法实现：



## ３.３可视化界面(MainWindow 类)

项目的可视化界面基于 python qt5 实现，可视化界面的设计使用 python qt5 的 Qt Designer 工具完成，并生成 ui 的 python 文件。

这里不做具体分析，我使用了Qlabel，Qpushbutton，VerticalLayoutWigde，CentralWidget等等进行了整个页面的开发，并且在进行过程中修改了字体等前端美化可视化。

## ３.４负责调用管理器、更新界面、响应用户输入的程序主逻辑部分(MyThread 类)。

　　实现了一个基于 Qt 框架的多线程模拟程序。具体功能如下:

MyThread 类继承自 QThread，实现了一个线程类。

在线程中有两个信号:

* my\_trigger\_update: 用于更新界面显示的各种参数值
* my\_trigger\_clear: 用于清空界面

checkReset 函数用于检查用户是否点击了重置按钮,如果点击了则清空界面并退出当前任务。

waitingFirstModeSet 和 waitingModeSet 函数用于等待用户输入执行模式,支持单步执行和连续执行两种模式。

run 函数是线程的主体,包含以下步骤:

* 初始化运行模式并等待用户第一次输入以启动模拟
* 初始化全局变量,如页面大小、代码数量等
* 初始化任务和管理器对象
* 进入任务级循环
* 检查是否设定了重置
* 等待用户设定执行模式
* 根据设定的调页算法执行任务代码,获取执行情况和调页情况
* 更新界面
* 果任务代码全部执行完毕,设定重置标志



def run(self):

while True: # 模拟级循环

# 初始化运行模式 准备读取用户的第一次输入以启动模拟

global run\_mode

p(run\_mode\_sem)

run\_mode = 0

v(run\_mode\_sem)

# 等待用户第一次输入执行方式以开始模拟

self.waitingFirstModeSet()

# 初始化全局变量 这部分不可以放到上面的waitingFirstModeSet的前面执行

# 上方的waitingFirstModeSet中，在等待用户输入的同时，用户可能点击reset按钮，所以必须在这里将reset重置

global reset\_flag

p(reset\_flag\_sem)

reset\_flag = 0

v(reset\_flag\_sem)

# 模拟参数初始化

page\_size = 10 # 一个页面可以存放的指令数

code\_amount = 320 # 从用户输入读取任务代码数

page\_swap\_algo = ui.comboBox.currentText() # 从用户输入读取所选算法

# 任务初始化

task = Task(page\_size=page\_size, code\_amount=code\_amount)

# 管理器初始化 传入不同的算法参数 则后续调用runTask时执行不同的算法

manager = MyManager(page\_size=page\_size, algo=page\_swap\_algo)

# 缺页数初始化

page\_missing = 0

# 初始化用于退出两层循环的标志

self.tmp\_reset\_flag = False

while True: # 任务级循环

# 检查是否设定了重置

if self.checkReset():

break

# 等待用户设定模式

self.waitingModeSet()

if self.tmp\_reset\_flag: # 上方waitingModeSet中读取到用户设定了reset

self.tmp\_reset\_flag = False

# 回到任务级循环的头部 由头部进行界面清空并退出任务级循环

# 如果将continue改成break，则会回到模拟级循环的头部，此时reset\_flag会被重新初始化成0，则任务级循环的头部无法进行界面的清空

continue

# 根据设定的调页算法进行Task代码执行，并进行内存分配或页面调换，并获取Task代码执行情况与调页情况

log\_code\_num, log\_cur\_code, log\_page\_missing, log\_old\_page, log\_code\_page, log\_memory\_page = manager.runTask(task)

if log\_page\_missing: # 发生缺页执行了换页

page\_missing = page\_missing + 1

if log\_code\_num >= (code\_amount - 1): # 运行完设定的Task的所有代码

p(reset\_flag\_sem)

reset\_flag = 1 # 设定重置，执行完下面的界面更新回到任务级循环后即由头部执行页面清空并退出任务级循环

v(reset\_flag\_sem)

# 更新界面

self.my\_trigger\_update.emit(log\_code\_num, log\_cur\_code, log\_page\_missing, log\_old\_page, log\_code\_page, log\_memory\_page, page\_missing, page\_size, code\_amount)

time.sleep(0.1)

# 文件架构以及运行

上交文件包括：

**源代码：**

Main.py

Managing,py

Task.py

Window.py-----Window.ui(pydesign进行开发，通过pyuic进行转换)

Main.spec（打包过程当中的源代码）

虚拟环境的.venv文件未上传

**打包exe可执行文件：**

dist文件内部，一个exe可执行文件，一个压缩zip版防止不可用

**说明文档：**

2251657\_李若冰\_项目说明文档.docx

**运行：**

若想本地运行，则需配置qt虚拟环境，运用python3.8进行本地运行

若想看项目展示，则直接进行exe即可。