江苏大学京江学院

JIANGSU UNIVERSITY JINGJIANG COLLEGE

操作系统课程设计

CPU & 内存调度管理原型系统

专业班级：	J软件(嵌入)(专转本)2102
学　　号：	4211153047
学生姓名：	马云骥
指导教师：	潘雨青
日　　期：	2024.01.10

操作系统课程设计

1 课设目的

操作系统课程设计是计算机科学与技术专业学生必修的实践性教学环节之一，是学习了操作系统课程后的综合性设计实践课程，是对该课程所学知识进行的一次全面的综合训练。通过学生完成所要求的设计任务，使学生系统掌握操作系统的基本原理，系统的设计与实现方法，培养学生利用所学知识解决复杂工程问题的能力。通过查阅资料、自学、指导和讨论，使学生掌握操作系统的功能模块的设计与实现方法；通过制定合理的实验方案和实验结果的分析，培养学生的科学实验能力；通过对实验结果的分析和总结以及课程设计报告的撰写，掌握科学实验方法以及科学实验报告的撰写方法；通过交流和答辩，培养学生的口头交流和表达能力。

2 课设题目

实现一个 CPU & 内存调度管理原型系统

3 系统功能结构

3.1 概述

本系统主要包含以下几个功能模块：

1. **进程管理**：负责创建和调度进程，管理进程的生命周期。
2. **页面调度**：处理内存分配和页面置换，以优化内存使用。
3. **结果记录**：记录和计算进程的周转时间和带权周转时间，输出到文件。
4. **用户交互**：提供用户界面，允许用户设置参数，如时间片长度、页面大小等。

3.2 模块间关系和流程

1. **用户交互**：系统启动时，用户通过菜单设置参数，这些参数影响进程管理和页面调度的行为。
2. **进程管理**：
   - 使用先来先服务（FCFS）或时间片轮转（RR）算法调度进程。
   - 管理进程状态（如就绪、运行、等待、完成）。
   - 与页面调度模块交互，请求必要的内存页面。
3. **页面调度**：
   - 根据进程管理模块的需求分配和回收内存页面。
   - 使用先进先出（FIFO）或最近最少使用（LRU）算法管理页面置换。
   - 直接影响进程的运行，尤其是在内存密集型操作中。
4. **结果记录**：

- 在进程完成后，计算其周转时间和带权周转时间。
- 将统计结果输出到文件 result.txt，以供分析和查阅。

4 主要数据结构

4.1 进程控制块（PCB）

pcb 类是一个关键的数据结构，代表了系统中的一个进程。它包含了进程的多种属性和状态信息，这些属性包括但不限于：

- name：进程名，用于标识每个独立的进程。
- status：进程的当前状态，如“就绪”、“运行”、“等待”等。
- arraiveTime：进程到达时间，即进程创建或提交执行的时间。
- priority：进程优先级，用于某些调度算法中判断进程的执行顺序。
- pageFrame：分配给进程的页面列表，用于内存管理。
- pageFaultCount：进程发生的缺页次数。
- serveTime 和 serveTimeLeft：进程的总服务时间和剩余服务时间。
- finishTime、turnAroundTime 和 weightTunAroundTime：分别记录进程的完成时间、周转时间和带权周转时间。
- fc：存储进程中的函数列表。
- runInfo：存储进程运行的关键信息，如运行的操作和时间节点。
- currentAddr：存储进程地址的指针，用于确定当前运行函数。

4.2 函数类

在 pcb 类中，function 类用于表示进程中的一个函数。它包含以下属性：

- name：函数名，标识函数。
- lenth：函数的大小（通常以字节为单位）。
- rtAddr：函数在进程中的相对地址。

这个类帮助管理和跟踪进程中的单独函数，对于页面调度和内存管理非常重要。

4.3 运行类

同样在 pcb 类中，run 类用于记录进程运行期间的关键事件。它包含以下信息：

- timeNode：事件发生的时间节点。
- operation：执行的操作类型，如“跳转”、“读写磁盘”或“结束”。
- operateAddr 和 ioTime：操作相关的地址或I/O操作时间。

- finishflag：标识操作是否完成。

4.4 页面列表

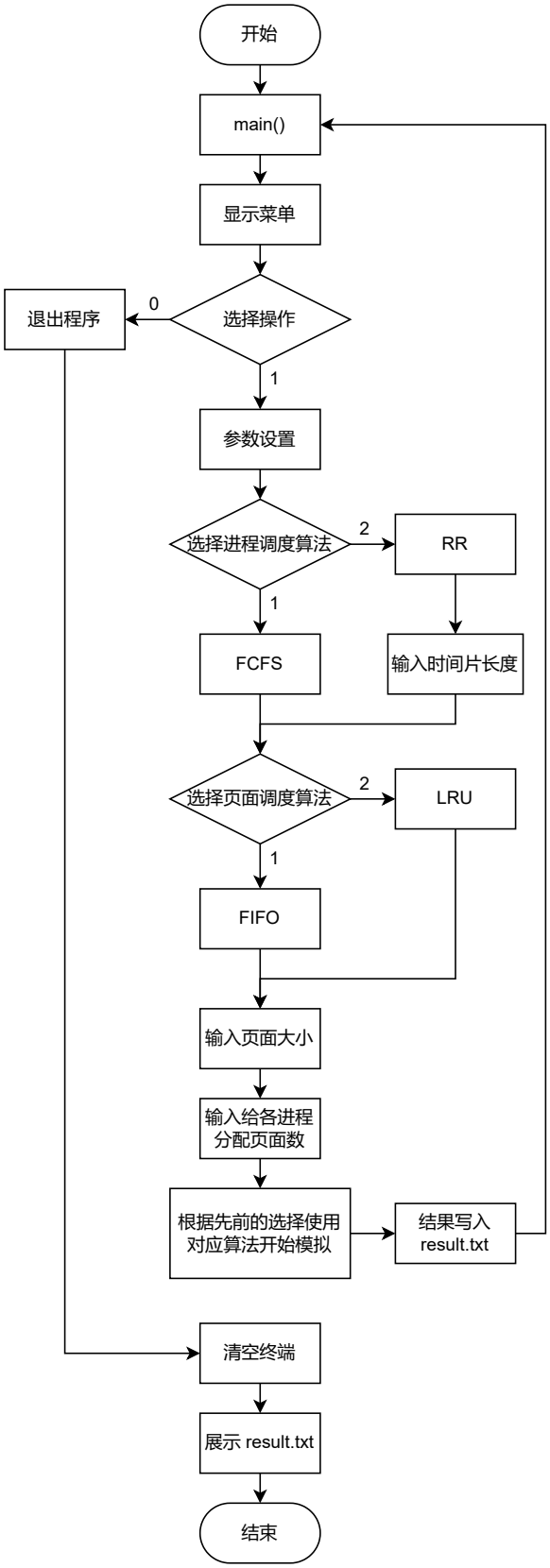
每个 pcb 对象都包含一个 pageFrame 列表，它代表为该进程分配的内存页面。这是一个关键的数据结构，用于管理每个进程在内存中的页面。

4.5 全局变量和队列

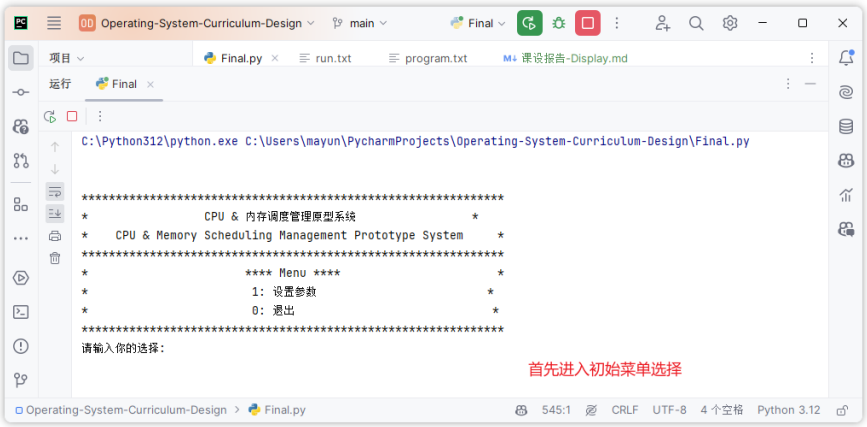
- readyQueue 和 ioQueue：分别用于管理就绪状态和等待I/O操作的进程队列。
- process：存储所有进程的列表。
- pageFrameAmount：记录为每个进程分配的页面数量。

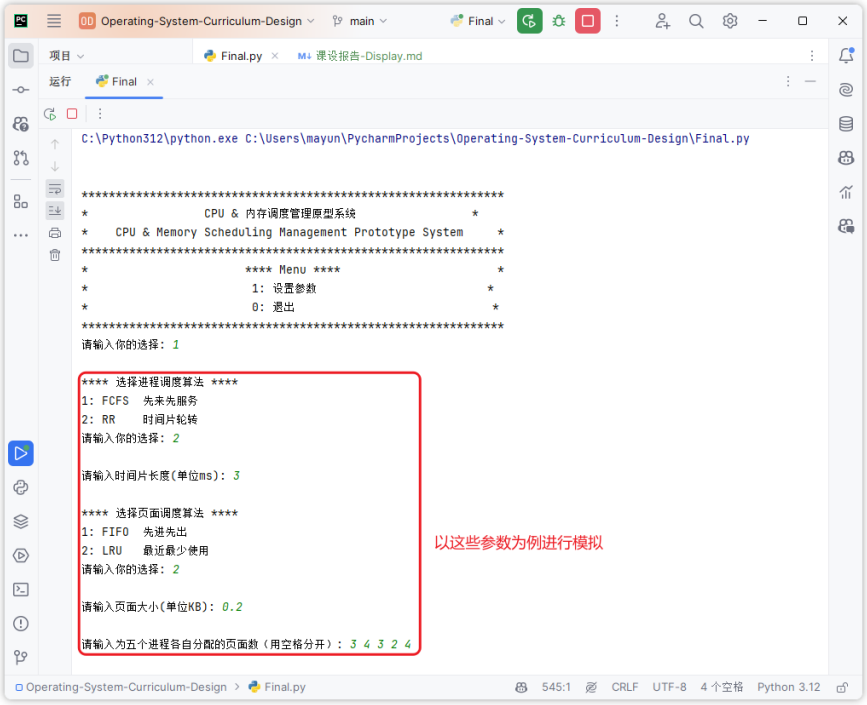
这些数据结构是实现进程调度和内存管理的基础，它们帮助本系统有效地跟踪和控制多个进程及其资源需求。

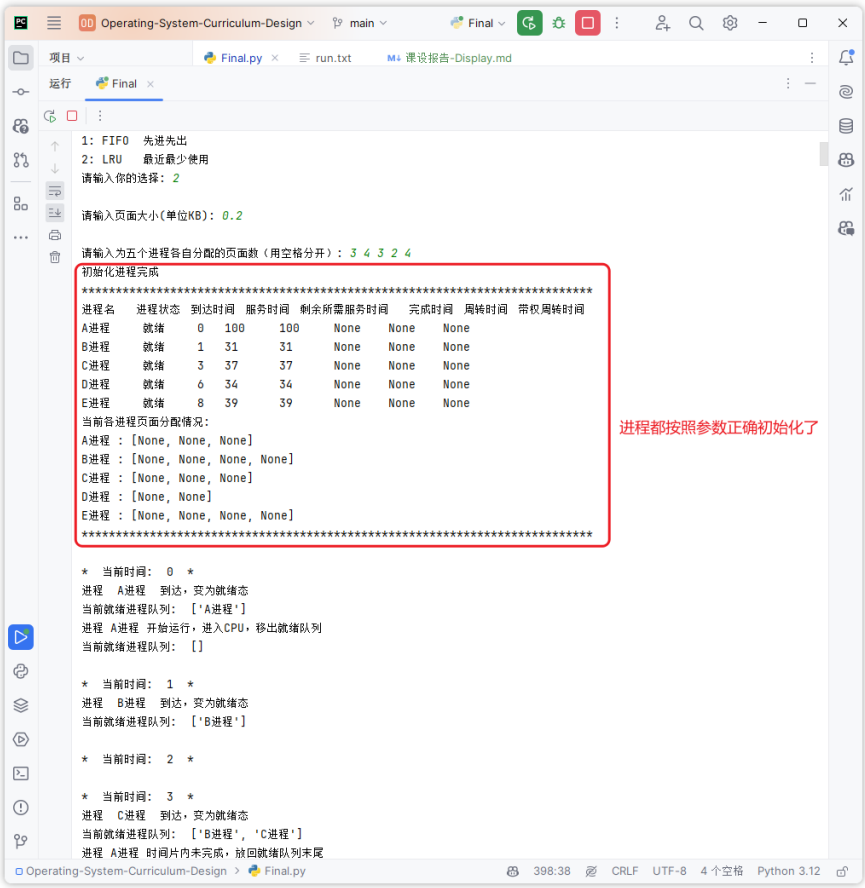
5 系统设计

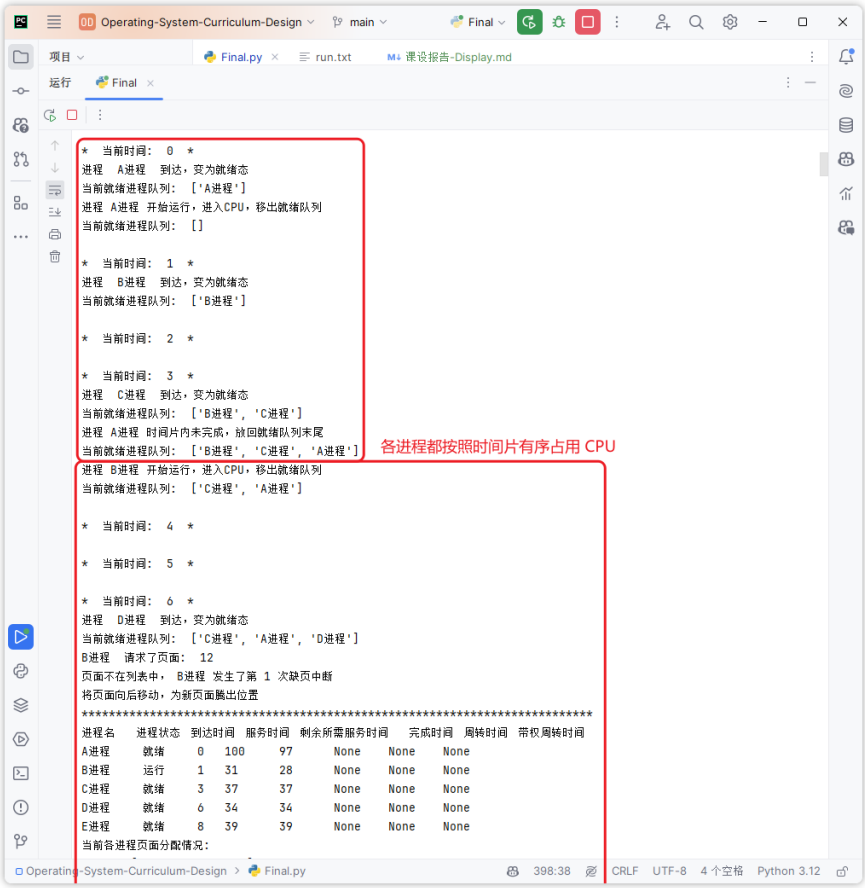

开始
main()
显示菜单
选择操作
0
退出程序
1
参数设置
选择进程调度算法
2
RR
1
FCFS
输入时间片长度
选择页面调度算法
2
LRU
1
FIFO
输入页面大小
输入给各进程
分配页面数
根据先前的选择使用
对应算法开始模拟
结果写入
result.txt
清空终端
展示 result.txt
结束

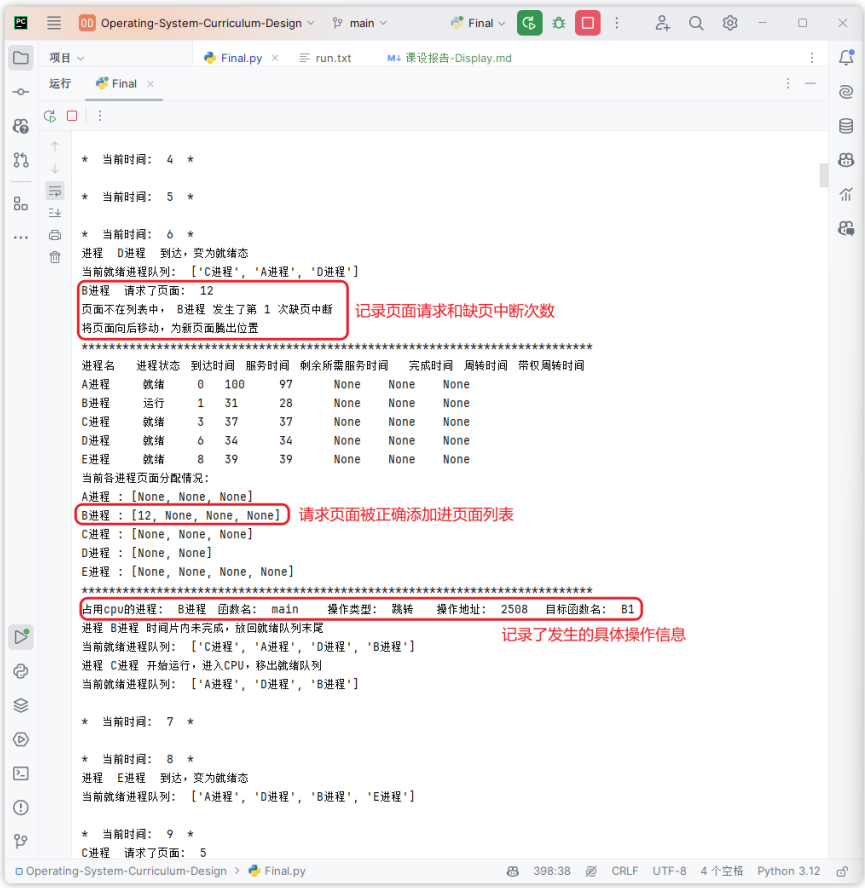
6 运行结果展示与分析

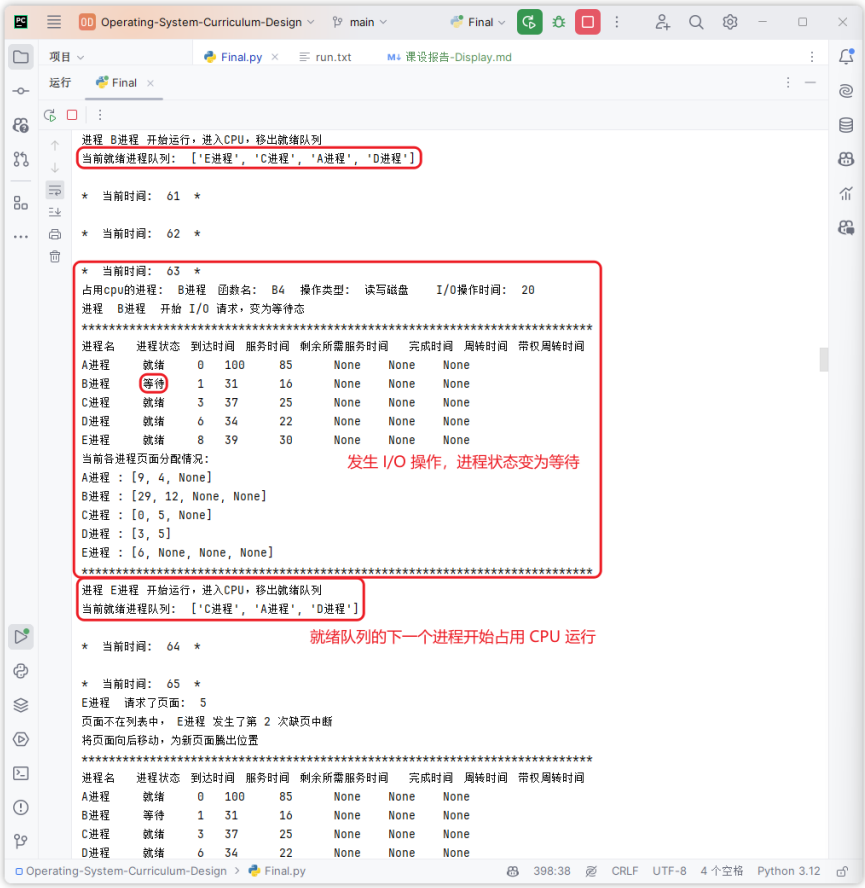

Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
program.txt
课设报告-Display.md
运行
Final
C:\Python312\python.exe C:\Users\mayun\PycharmProjects\Operating-System-Curriculum-Design\Final.py
****
* CPU & 内存调度管理原型系统 *
* CPU & Memory Scheduling Management Prototype System *
****
* **** Menu **** *
* 1: 设置参数 *
* 0: 退出 *
****
请输入你的选择:
首先进入初始菜单选择
Operating-System-Curriculum-Design
Final.py
545:1
CRLF
UTF-8
4 个空格
Python 3.12

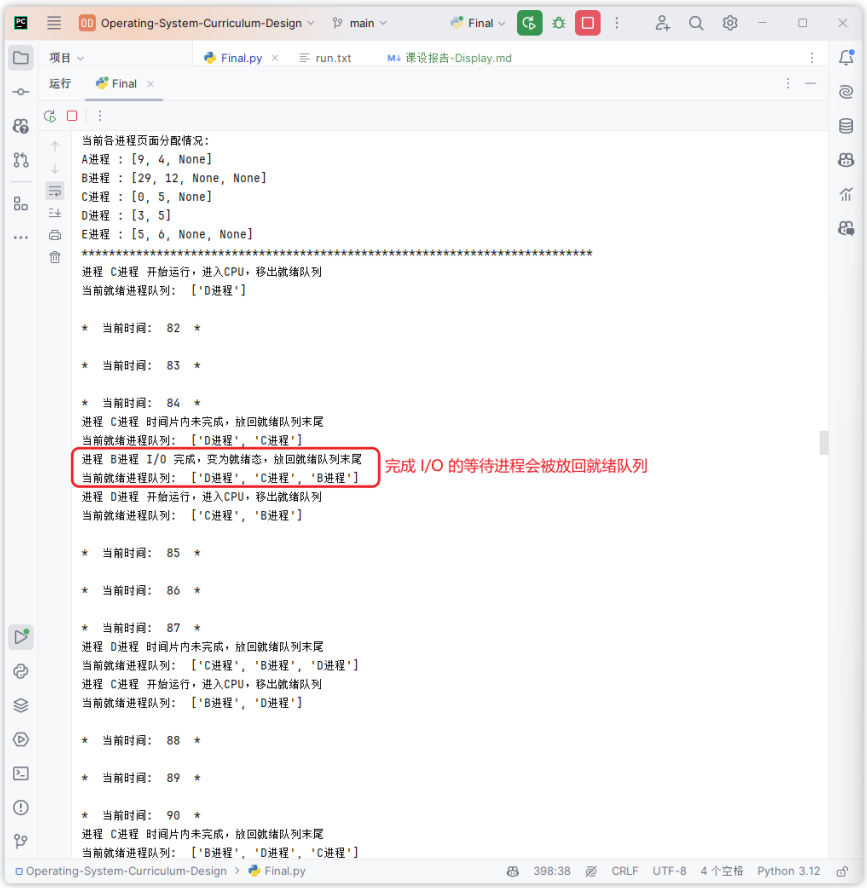

Operating-System-Curriculum-Design
main
Final
项目
Final.py
课设报告-Display.md
运行
Final
C:\Python312\python.exe C:\Users\mayun\PycharmProjects\Operating-System-Curriculum-Design\Final.py
****
* CPU & 内存调度管理原型系统 *
* CPU & Memory Scheduling Management Prototype System *
****
* **** Menu **** *
* 1: 设置参数 *
* 0: 退出 *
****
请输入你的选择: 1
**** 选择进程调度算法 ****
1: FCFS 先来先服务
2: RR 时间片轮转
请输入你的选择: 2
请输入时间片长度(单位ms): 3
**** 选择页面调度算法 ****
1: FIFO 先进先出
2: LRU 最近最少使用
请输入你的选择: 2
请输入页面大小(单位KB): 0.2
请输入为五个进程各自分配的页面数(用空格分开): 3 4 3 2 4
以这些参数为例进行模拟
Operating-System-Curriculum-Design
Final.py
545:1
CRLF
UTF-8
4 个空格
Python 3.12

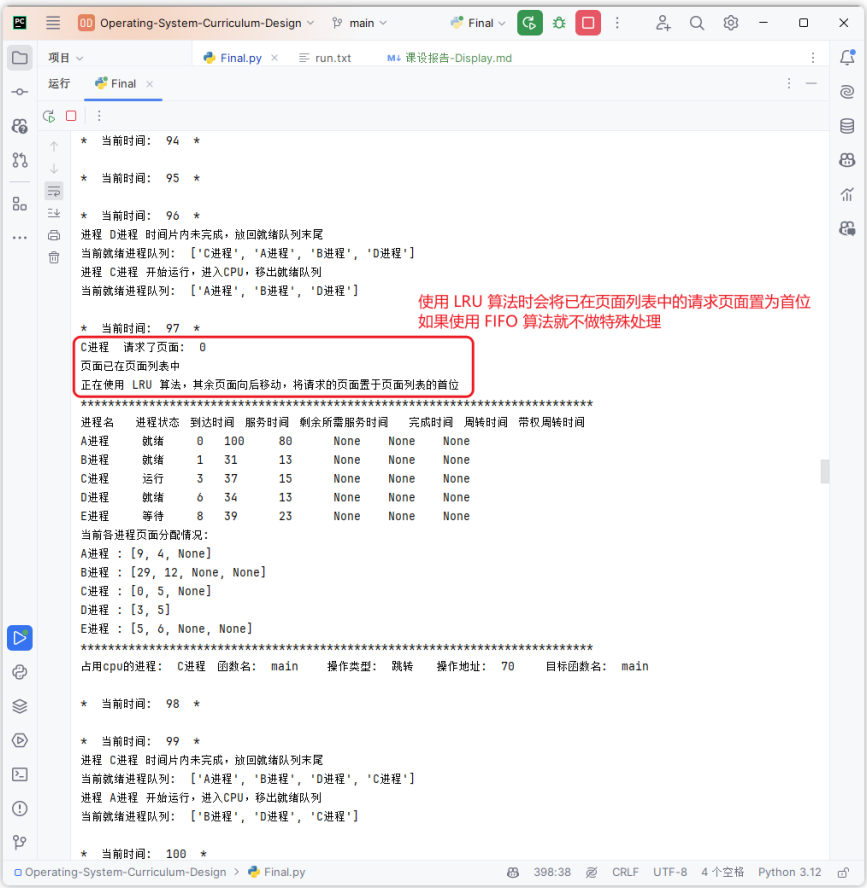

Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
1: FIFO 先进先出
2: LRU 最近最少使用
请输入你的选择: 2
请输入页面大小(单位KB): 0.2
请输入为五个进程各自分配的页面数（用空格分开）: 3 4 3 2 4
初始化进程完成
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 100 None None None
B进程 就绪 1 31 31 None None None
C进程 就绪 3 37 37 None None None
D进程 就绪 6 34 34 None None None
E进程 就绪 8 39 39 None None None
当前各进程页面分配情况:
A进程 : [None, None, None]
B进程 : [None, None, None, None]
C进程 : [None, None, None]
D进程 : [None, None]
E进程 : [None, None, None, None]
进程都按照参数正确初始化了
* 当前时间: 0 *
进程 A进程 到达，变为就绪态
当前就绪进程队列: ['A进程']
进程 A进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: []
* 当前时间: 1 *
进程 B进程 到达，变为就绪态
当前就绪进程队列: ['B进程']
* 当前时间: 2 *
* 当前时间: 3 *
进程 C进程 到达，变为就绪态
当前就绪进程队列: ['B进程', 'C进程']
进程 A进程 时间片内未完成，放回就绪队列末尾
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12

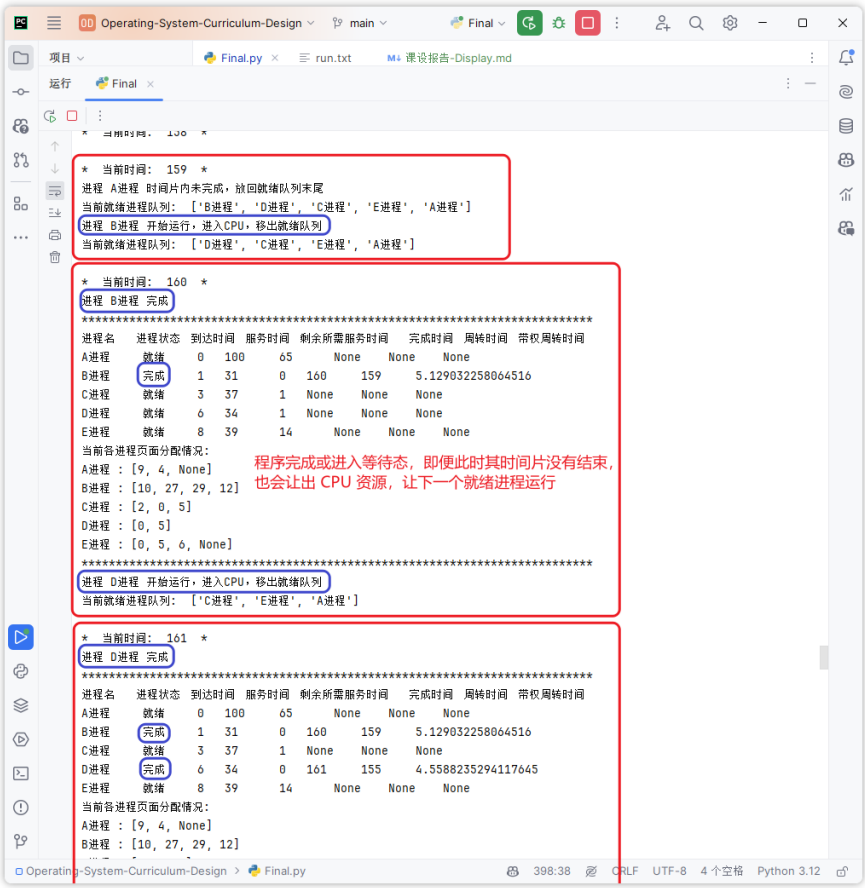

Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
* 当前时间: 0 *
进程 A进程 到达，变为就绪态
当前就绪进程队列: ['A进程']
进程 A进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: []
* 当前时间: 1 *
进程 B进程 到达，变为就绪态
当前就绪进程队列: ['B进程']
* 当前时间: 2 *
* 当前时间: 3 *
进程 C进程 到达，变为就绪态
当前就绪进程队列: ['B进程', 'C进程']
进程 A进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['B进程', 'C进程', 'A进程']
各进程都按照时间片有序占用 CPU
进程 B进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['C进程', 'A进程']
* 当前时间: 4 *
* 当前时间: 5 *
* 当前时间: 6 *
进程 D进程 到达，变为就绪态
当前就绪进程队列: ['C进程', 'A进程', 'D进程']
B进程 请求了页面: 12
页面不在列表中， B进程 发生了第 1 次缺页中断
将页面向后移动，为新页面腾出位置
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 97 None None None
B进程 运行 1 31 28 None None None
C进程 就绪 3 37 37 None None None
D进程 就绪 6 34 34 None None None
E进程 就绪 8 39 39 None None None
当前各进程页面分配情况:
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12

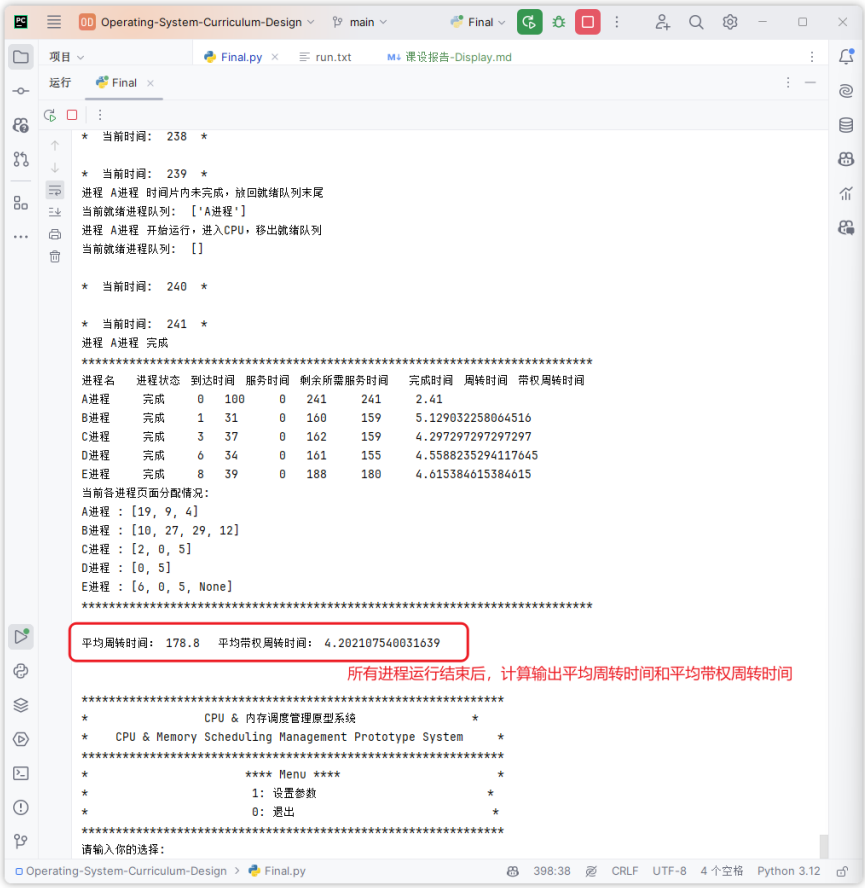

Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
* 当前时间: 4 *
* 当前时间: 5 *
* 当前时间: 6 *
进程 D进程 到达，变为就绪态
当前就绪进程队列: ['C进程', 'A进程', 'D进程']
B进程 请求了页面: 12
页面不在列表中， B进程 发生了第 1 次缺页中断
将页面向后移动，为新页面腾出位置
记录页面请求和缺页中断次数
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 97 None None None
B进程 运行 1 31 28 None None None
C进程 就绪 3 37 37 None None None
D进程 就绪 6 34 34 None None None
E进程 就绪 8 39 39 None None None
当前各进程页面分配情况:
A进程 : [None, None, None]
B进程 : [12, None, None, None]
请求页面被正确添加进页面列表
C进程 : [None, None, None]
D进程 : [None, None]
E进程 : [None, None, None, None]
占用cpu的进程: B进程 函数名: main 操作类型: 跳转 操作地址: 2508 目标函数名: B1
记录了发生的具体操作信息
进程 B进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['C进程', 'A进程', 'D进程', 'B进程']
进程 C进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['A进程', 'D进程', 'B进程']
* 当前时间: 7 *
* 当前时间: 8 *
进程 E进程 到达，变为就绪态
当前就绪进程队列: ['A进程', 'D进程', 'B进程', 'E进程']
* 当前时间: 9 *
C进程 请求了页面: 5
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12

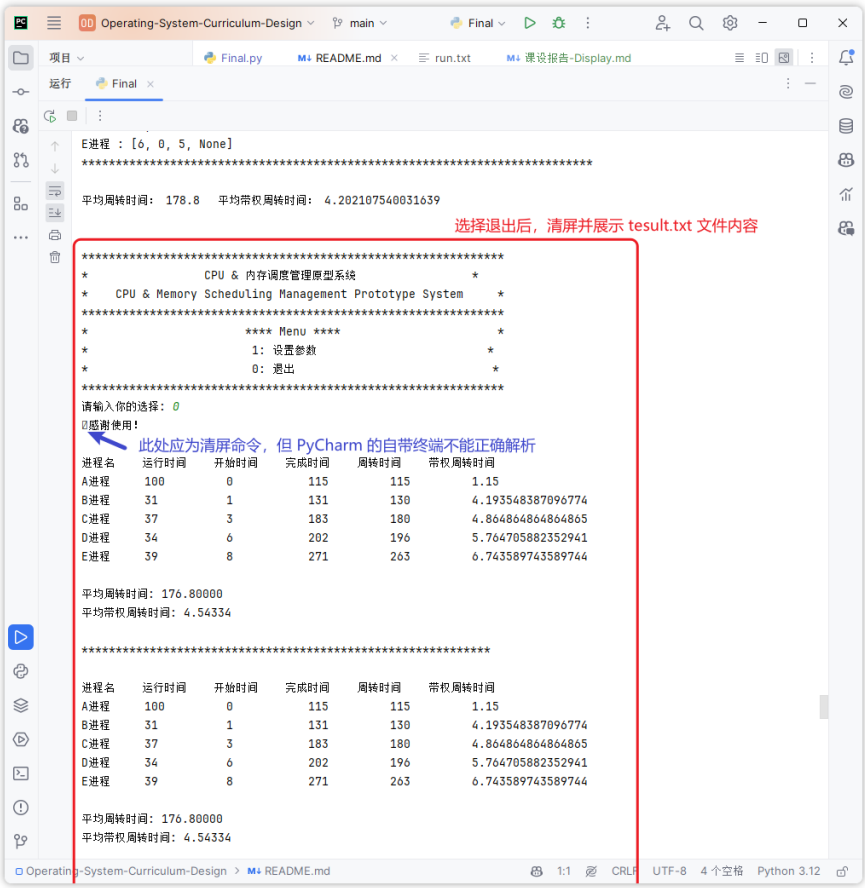

Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
进程 B进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['E进程', 'C进程', 'A进程', 'D进程']
* 当前时间: 61 *
* 当前时间: 62 *
* 当前时间: 63 *
占用cpu的进程: B进程 函数名: B4 操作类型: 读写磁盘 I/O操作时间: 20
进程 B进程 开始 I/O 请求，变为等待态
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 85 None None None
B进程 等待 1 31 16 None None None
C进程 就绪 3 37 25 None None None
D进程 就绪 6 34 22 None None None
E进程 就绪 8 39 30 None None None
当前各进程页面分配情况:
发生 I/O 操作，进程状态变为等待
A进程 : [9, 4, None]
B进程 : [29, 12, None, None]
C进程 : [0, 5, None]
D进程 : [3, 5]
E进程 : [6, None, None, None]
进程 E进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['C进程', 'A进程', 'D进程']
就绪队列的下一个进程开始占用 CPU 运行
* 当前时间: 64 *
* 当前时间: 65 *
E进程 请求了页面: 5
页面不在列表中， E进程 发生了第 2 次缺页中断
将页面向后移动，为新页面腾出位置
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 85 None None None
B进程 等待 1 31 16 None None None
C进程 就绪 3 37 25 None None None
D进程 就绪 6 34 22 None None None
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12


Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
当前各进程页面分配情况:
A进程 : [9, 4, None]
B进程 : [29, 12, None, None]
C进程 : [0, 5, None]
D进程 : [3, 5]
E进程 : [5, 6, None, None]
进程 C进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['D进程']
* 当前时间: 82 *
* 当前时间: 83 *
* 当前时间: 84 *
进程 C进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['D进程', 'C进程']
进程 B进程 I/O 完成，变为就绪态，放回就绪队列末尾
当前就绪进程队列: ['D进程', 'C进程', 'B进程']
完成 I/O 的等待进程会被放回就绪队列
进程 D进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['C进程', 'B进程']
* 当前时间: 85 *
* 当前时间: 86 *
* 当前时间: 87 *
进程 D进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['C进程', 'B进程', 'D进程']
进程 C进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['B进程', 'D进程']
* 当前时间: 88 *
* 当前时间: 89 *
* 当前时间: 90 *
进程 C进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['B进程', 'D进程', 'C进程']
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12


Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
* 当前时间: 94 *
* 当前时间: 95 *
* 当前时间: 96 *
进程 D进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['C进程', 'A进程', 'B进程', 'D进程']
进程 C进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['A进程', 'B进程', 'D进程']
使用 LRU 算法时会将已在页面列表中的请求页面置为首位
如果使用 FIFO 算法就不做特殊处理
* 当前时间: 97 *
C进程 请求了页面: 0
页面已在页面列表中
正在使用 LRU 算法，其余页面向后移动，将请求的页面置于页面列表的首位
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 80 None None None
B进程 就绪 1 31 13 None None None
C进程 运行 3 37 15 None None None
D进程 就绪 6 34 13 None None None
E进程 等待 8 39 23 None None None
当前各进程页面分配情况:
A进程 : [9, 4, None]
B进程 : [29, 12, None, None]
C进程 : [0, 5, None]
D进程 : [3, 5]
E进程 : [5, 6, None, None]
占用cpu的进程: C进程 函数名: main 操作类型: 跳转 操作地址: 70 目标函数名: main
* 当前时间: 98 *
* 当前时间: 99 *
进程 C进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['A进程', 'B进程', 'D进程', 'C进程']
进程 A进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['B进程', 'D进程', 'C进程']
* 当前时间: 100 *
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12


Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
* 当前时间: 159 *
进程 A进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['B进程', 'D进程', 'C进程', 'E进程', 'A进程']
进程 B进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['D进程', 'C进程', 'E进程', 'A进程']
* 当前时间: 160 *
进程 B进程 完成
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 65 None None None
B进程 完成 1 31 0 160 159 5.129032258064516
C进程 就绪 3 37 1 None None None
D进程 就绪 6 34 1 None None None
E进程 就绪 8 39 14 None None None
当前各进程页面分配情况:
A进程 : [9, 4, None]
B进程 : [10, 27, 29, 12]
C进程 : [2, 0, 5]
D进程 : [0, 5]
E进程 : [0, 5, 6, None]
进程 D进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: ['C进程', 'E进程', 'A进程']
程序完成或进入等待态，即便此时其时间片没有结束，也会让出 CPU 资源，让下一个就绪进程运行
* 当前时间: 161 *
进程 D进程 完成
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 就绪 0 100 65 None None None
B进程 完成 1 31 0 160 159 5.129032258064516
C进程 就绪 3 37 1 None None None
D进程 完成 6 34 0 161 155 4.5588235294117645
E进程 就绪 8 39 14 None None None
当前各进程页面分配情况:
A进程 : [9, 4, None]
B进程 : [10, 27, 29, 12]
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12


Operating-System-Curriculum-Design
main
Final
项目
Final.py
run.txt
课设报告-Display.md
运行
Final
* 当前时间: 238 *
* 当前时间: 239 *
进程 A进程 时间片内未完成，放回就绪队列末尾
当前就绪进程队列: ['A进程']
进程 A进程 开始运行，进入CPU，移出就绪队列
当前就绪进程队列: []
* 当前时间: 240 *
* 当前时间: 241 *
进程 A进程 完成
进程名 进程状态 到达时间 服务时间 剩余所需服务时间 完成时间 周转时间 带权周转时间
A进程 完成 0 100 0 241 241 2.41
B进程 完成 1 31 0 160 159 5.129032258064516
C进程 完成 3 37 0 162 159 4.297297297297297
D进程 完成 6 34 0 161 155 4.5588235294117645
E进程 完成 8 39 0 188 180 4.615384615384615
当前各进程页面分配情况:
A进程 : [19, 9, 4]
B进程 : [10, 27, 29, 12]
C进程 : [2, 0, 5]
D进程 : [0, 5]
E进程 : [6, 0, 5, None]
平均周转时间: 178.8 平均带权周转时间: 4.202107540031639
所有进程运行结束后，计算输出平均周转时间和平均带权周转时间
CPU & 内存调度管理原型系统
CPU & Memory Scheduling Management Prototype System
**** Menu ****
1: 设置参数
0: 退出
请输入你的选择:
Operating-System-Curriculum-Design > Final.py
398:38 CRLF UTF-8 4 个空格 Python 3.12



7 课程设计中遇到的问题及解决方法

实现先来先服务和时间片轮转调度算法时，需要考虑进程状态的管理和时间的精确控制，这比我预期的设想更复杂。通过仔细设计和维护进程状态，确保在每个时间点正确处理进程。例如，使用队列来管理就绪状态的进程，并使用计时器或循环计数来模拟时间片的流逝。

设计一个易于使用且直观的用户界面，使用户能够轻松设置参数和查看程序运行结果，这也是一个挑战。最初打算使用 Python 标准库中的 tkinter 实现一个可视化的界面，但时间已经不允许我从零学习和实现了。最终我选择采用简单但有效的命令行界面设计，确保清晰的指令和输出信息。

8 设计感想

在完成这个操作系统课程设计的过程中，我获得了不仅是编程技能的提升，更重要的是对操作系统原理和内部工作机制的深入理解。这个项目不仅是对我所学知识的实践应用，也是对我解决问题能力的重大考验。

通过这个项目，我意识到理论知识和实践操作之间的重要联系。虽然理论课程提供了必要的基础，但将这些理论应用到实际问题中，需要一种全新的思考方式。在实现进程调度和页面管理算法时，我不得不反复回顾理论，确保我对概念的理解是准确的。

这个项目中我选择了使用 Python 来实现较为复杂的逻辑。在这个过程中，我的编程能力得到了显著提升。我学会了如何更有效地组织代码，使其既易于理解又易于维护。同时，我也学会了一些高级编程技巧，比如面向对象编程和对 Python List 列表数据结构的高效使用。

通过这次课程设计，我对操作系统的工作原理有了更深的理解，也对计算机科学领域的其他方面产生了浓厚的兴趣。我期待在未来的学习和职业生涯中，将这次的学习经历转化为更大的成功。

9 附件

```
import math
import os
import platform
import time

mainflag = None  # 用于控制主菜单的循环
cpuflag = None  # 用于控制进程调度算法的选择
memoryflag = None  # 用于控制页面调度算法的选择
timeSlice = None  # 时间片长度
pageLength = None  # 单页面大小
TIME = None  # 单位时间
process = None  # 进程对象列表
pageFrameAmount = None  # 页面数
ioQueue = None  # I/O请求队列
readyQueue = None  # 就绪队列
avgTurnAroundTime = None  # 平均周转时间
avgWeightTurnAroundTime = None  # 平均带权周转时间
running_process = None  # 当前运行的进程


class pcb:
    def __init__(self, name, status, arraiveTime, priority, pageFrame, pageFaultCount, serveTime, serveTimeLeft,
                 finishTime, turnAroundTime, weightTunAroundTime, fc, runInfo, iostart, iofinish, currentAddr):
        self.name = name  # 进程名
        self.status = status  # 进程状态
        self.arraiveTime = arraiveTime  # 创建时间
        self.priority = priority  # 优先级
        self.pageFrame = pageFrame  # 页面列表
        self.pageFaultCount = pageFaultCount  # 缺页次数
        self.serveTime = serveTime  # 服务时间
        self.serveTimeLeft = serveTimeLeft  # 剩余所需服务时间
        self.finishTime = finishTime  # 完成时间
        self.turnAroundTime = turnAroundTime  # 周转时间
        self.weightTunAroundTime = weightTunAroundTime  # 带权周转时间
        self.fc = fc
        self.runInfo = runInfo
        self.iostart = iostart  # I/O开始时间
        self.iofinish = iofinish  # I/O结束时间
        self.currentAddr = currentAddr  # 当前操作地址

    class function:
        def __init__(self, name, lenth, rtAddr):
            self.name = name  # 文件名
            self.lenth = lenth  # 大小
            self.rtAddr = rtAddr  # 相对地址

    class run:
        def __init__(self, timeNode, operation, operateAddr, ioTime, finishflag):
            self.timeNode = timeNode  # 时间节点
            self.operation = operation  # 操作名称
```

```
            self.operateAddr = operateAddr  # 操作地址
            self.ioTime = ioTime  # I/O操作时间
            self.finishflag = finishflag  # 结束标志


def create():
    global process, pageFrameAmount

    # 直接将三个txt文件的内容按格式硬编码为list对象
    process = []
    process.append(pcb('A进程', '就绪', 0, 5, None, 0, 100, 100, None, None, None, None, None, None, None, 0))
    process.append(pcb('B进程', '就绪', 1, 4, None, 0, 31, 31, None, None, None, None, None, None, None, 0))
    process.append(pcb('C进程', '就绪', 3, 7, None, 0, 37, 37, None, None, None, None, None, None, None, 0))
    process.append(pcb('D进程', '就绪', 6, 5, None, 0, 34, 34, None, None, None, None, None, None, None, 0))
    process.append(pcb('E进程', '就绪', 8, 6, None, 0, 39, 39, None, None, None, None, None, None, None, 0))

    for p in process:
        p.pageFrame = []
        for i in range(pageFrameAmount[process.index(p)]):
            p.pageFrame.append(None)

    process[0].fc = []
    process[0].fc.append(pcb.function('main', 0.6, None))
    process[0].fc.append(pcb.function('A1', 1.2, None))
    process[0].fc.append(pcb.function('A2', 1.2, None))
    process[0].fc.append(pcb.function('A3', 1.5, None))
    process[0].fc.append(pcb.function('A4', 0.8, None))

    process[1].fc = []
    process[1].fc.append(pcb.function('main', 1.6, None))
    process[1].fc.append(pcb.function('B1', 2.2, None))
    process[1].fc.append(pcb.function('B2', 0.2, None))
    process[1].fc.append(pcb.function('B3', 0.5, None))
    process[1].fc.append(pcb.function('B4', 1.8, None))
    process[1].fc.append(pcb.function('B5', 0.9, None))

    process[2].fc = []
    process[2].fc.append(pcb.function('main', 0.3, None))
    process[2].fc.append(pcb.function('C1', 0.1, None))
    process[2].fc.append(pcb.function('C2', 0.3, None))
    process[2].fc.append(pcb.function('C3', 0.5, None))

    process[3].fc = []
    process[3].fc.append(pcb.function('main', 0.9, None))
    process[3].fc.append(pcb.function('D1', 1.6, None))
    process[3].fc.append(pcb.function('D2', 1.8, None))
    process[3].fc.append(pcb.function('D3', 2.0, None))
    process[3].fc.append(pcb.function('D4', 0.9, None))

    process[4].fc = []
    process[4].fc.append(pcb.function('main', 0.7, None))
```

```
    process[4].fc.append(pcb.function('E1', 0.3, None))
    process[4].fc.append(pcb.function('E2', 0.5, None))
    process[4].fc.append(pcb.function('E3', 0.9, None))
    process[4].fc.append(pcb.function('E4', 0.3, None))

    for p in process:
        total = 0
        for f in p.fc:
            f.lenth = math.ceil(f.lenth * 1024)  # 将 KB 转换为 Byte，对不满 1B 的部分使用 math.ceil() 向上取整
            f.rtAddr = total
            total += f.lenth
        p.totalsize = total

    process[0].runInfo = []
    process[0].runInfo.append(pcb.run(5, '跳转', 1021, None, False))
    process[0].runInfo.append(pcb.run(10, '跳转', 2021, None, False))
    process[0].runInfo.append(pcb.run(20, '读写磁盘', None, 10, False))
    process[0].runInfo.append(pcb.run(30, '跳转', 2031, None, False))
    process[0].runInfo.append(pcb.run(70, '跳转', 4050, None, False))
    process[0].runInfo.append(pcb.run(100, '结束', None, None, False))

    process[1].runInfo = []
    process[1].runInfo.append(pcb.run(3, '跳转', 2508, None, False))
    process[1].runInfo.append(pcb.run(10, '跳转', 6007, None, False))
    process[1].runInfo.append(pcb.run(15, '读写磁盘', None, 20, False))
    process[1].runInfo.append(pcb.run(22, '跳转', 5737, None, False))
    process[1].runInfo.append(pcb.run(27, '跳转', 2245, None, False))
    process[1].runInfo.append(pcb.run(31, '结束', 6311, None, False))

    process[2].runInfo = []
    process[2].runInfo.append(pcb.run(3, '跳转', 1074, None, False))
    process[2].runInfo.append(pcb.run(9, '跳转', 94, None, False))
    process[2].runInfo.append(pcb.run(15, '读写磁盘', None, 10, False))
    process[2].runInfo.append(pcb.run(22, '跳转', 70, None, False))
    process[2].runInfo.append(pcb.run(30, '跳转', 516, None, False))
    process[2].runInfo.append(pcb.run(37, '结束', 50, None, False))

    process[3].runInfo = []
    process[3].runInfo.append(pcb.run(3, '跳转', 1037, None, False))
    process[3].runInfo.append(pcb.run(10, '跳转', 782, None, False))
    process[3].runInfo.append(pcb.run(15, '读写磁盘', None, 4, False))
    process[3].runInfo.append(pcb.run(22, '跳转', 1168, None, False))
    process[3].runInfo.append(pcb.run(28, '跳转', 79, None, False))
    process[3].runInfo.append(pcb.run(34, '结束', 431, None, False))

    process[4].runInfo = []
    process[4].runInfo.append(pcb.run(3, '跳转', 1414, None, False))
    process[4].runInfo.append(pcb.run(11, '跳转', 1074, None, False))
    process[4].runInfo.append(pcb.run(16, '读写磁盘', None, 30, False))
    process[4].runInfo.append(pcb.run(24, '跳转', 149, None, False))
    process[4].runInfo.append(pcb.run(32, '跳转', 1273, None, False))
    process[4].runInfo.append(pcb.run(39, '结束', 2053, None, False))

    readyList = []

```

```
157
158 # 显示进程信息
159 def showProcess():
160     print("********************************************************************************")
161     print("进程名\t进程状态\t到达时间\t服务时间\t剩余所需服务时间\t完成时间\t周转时间\t带权周
转时间")
162     for p in process:
163         print(p.name, '\t', p.status, '\t', p.arraiveTime, '\t', p.serveTime, '\t',
p.serveTimeLeft, '\t', p.finishTime,
164               '\t',
165               p.turnAroundTime, '\t', p.weightTunAroundTime)
166     print("当前各进程页面分配情况: ")
167     for p in process:
168         print(p.name, ":", p.pageFrame)
169     print("********************************************************************************")
170
171
172 # 显示当前就绪队列
173 def showReadyQueue(readyQueue):
174     show = []
175     for p in readyQueue:
176         show.append(p.name)
177     print("当前就绪进程队列: ", show)
178
179
180 # 页面置换
181 def replacePage(n, index):
182     global process
183     print("将页面向后移动，为新页面腾出位置")
184     if process[index].pageFrame[pageFrameAmount[index] - 1] is not None:
185         print("页面列表已满，将页面 ", process[index].pageFrame[pageFrameAmount[index] -
1], " 移出内存")
186     # 将页面向后移动，为新页面腾出位置
187     for i in range(pageFrameAmount[index] - 1, 0, -1):
188         process[index].pageFrame[i] = process[index].pageFrame[i - 1]
189     # 将新页面放置在页面列表的首位
190     process[index].pageFrame[0] = n
191
192
193 # 请求页面
194 def requirePage(addr, index):
195     global process, memoryflag
196     a = 0  # 帮助检查请求页面是否已在页面列表中的计数器
197     n = math.floor(addr / pageLength)  # 计算请求的页面号
198     print(process[index].name, " 请求了页面: ", n)
199
200     for i in range(pageFrameAmount[index]):
201         if process[index].pageFrame[i] == n:
202             print("页面已在页面列表中")
203             if memoryflag == '2':  # 如果使用LRU算法
204                 print("正在使用 LRU 算法，其余页面向后移动，将请求的页面置于页面列表的首位")
205                 # 将页面向后移动，将请求的页面置于页面列表的首位
206                 for j in range(i, 0, -1):
207                     process[index].pageFrame[j] = process[index].pageFrame[j - 1]
208                 process[index].pageFrame[0] = n
209             return n
```

```
        else:
            a += 1

    if a == pageFrameAmount[index]:
        process[index].pageFaultCount += 1
        print("页面不在列表中，", process[index].name, "发生了第",
process[index].pageFaultCount, "次缺页中断")
        replacePage(n, index)

    return n


def fcfs():
    global process, TIME, ioQueue, readyQueue, avgTurnAroundTime,
avgWeightTurnAroundTime, running_process

    create()  # 初始化进程
    print("初始化进程完成")
    showProcess()

    TIME = 0
    print("")
    print("* ", "当前时间: ", TIME, " *")
    time.sleep(0.1)
    ioQueue = []
    readyQueue = [p for p in process if p.arraiveTime <= TIME]
    print("进程 ", readyQueue[0].name, " 到达，变为就绪态")
    showReadyQueue(readyQueue)

    running_process = None  # 添加一个变量来追踪当前运行的进程

    while readyQueue or ioQueue or running_process:
        if running_process and running_process.serveTimeLeft == 0:
            # 如果当前运行的进程已经完成
            finish_process(running_process)
            running_process = None

            # 如果所有进程都已经完成
            if not (readyQueue or ioQueue):
                # 计算平均周转时间和带权周转时间
                calculate_averages()
                break

        check_io_queue()  # 检查IO队列中是否有完成IO的进程

        if not running_process and readyQueue:
            # 如果没有进程在运行，并且就绪队列不为空
            running_process = readyQueue.pop(0)
            start_process(running_process)
            showReadyQueue(readyQueue)

        # 运行当前进程一个时间单位
        TIME += 1
        if running_process:
            running_process.serveTimeLeft -= 1
        print("\n* ", "当前时间: ", TIME, " *")
```

```
        time.sleep(0.1)

        check_arriving_processes()  # 检查新到达的进程

        if running_process:
            if running_process.serveTimeLeft > 0:
                check_page_jump(running_process)  # 检查页面跳转
                if check_io_request(running_process):
                    # 如果有 IO 请求发生
                    running_process = None  # 暂停当前进程


def check_page_jump(proc):
    for r in proc.runInfo:
        if r.timeNode == proc.serveTime - proc.serveTimeLeft:
            if r.operation == '跳转':
                # 处理页面跳转
                requirePage(r.operateAddr, process.index(proc))
                showProcess()
                # 遍历proc.fc, 找到当前地址对应的函数名
                for f in proc.fc:
                    if f.rtAddr > proc.currentAddr:
                        break
                    currentFunctionName = f.name
                # 遍历proc.fc, 找到目标地址对应的函数名
                for f in proc.fc:
                    if f.rtAddr > r.operateAddr:
                        break
                    targetFunctionName = f.name
                # 更新进程的当前地址
                proc.currentAddr = r.operateAddr
                print("占用cpu的进程: ", proc.name, "\t函数名: ", currentFunctionName,
                      "\t操作类型: ", r.operation, "\t操作地址: ", r.operateAddr, "\t目标函
数名: ",
                      targetFunctionName)


def finish_process(proc):
    proc.status = '完成'
    print(f"进程 {proc.name} 完成")
    proc.finishTime = TIME
    proc.turnAroundTime = TIME - proc.arraiveTime
    proc.weightTunAroundTime = proc.turnAroundTime / proc.serveTime
    showProcess()


def check_io_queue():
    global TIME, running_process
    for p in list(ioQueue):
        if p.iofinish == TIME:
            print(f"进程 {p.name} I/O 完成, 变为就绪态, 放入就绪队列首部")
            p.status = '就绪'
            readyQueue.insert(0, p)
            showReadyQueue(readyQueue)
            ioQueue.remove(p)

```

```
            if running_process:
                # 如果当前有进程在运行，将其移回就绪队列
                print(f"暂停当前运行的进程 {running_process.name}，将其放在绪队列刚完成 I/O
的进程后面")
                running_process.status = '就绪'
                readyQueue.insert(1, running_process)
                showReadyQueue(readyQueue)

            running_process = readyQueue.pop(0)  # 将刚完成 I/O 的进程移出就绪队列
            print("刚完成 I/O 的进程 ", p.name, " 开始运行，进入CPU，移出就绪队列")
            showReadyQueue(readyQueue)
            return True
            break
    return False


def start_process(proc):
    print(f"进程 {proc.name} 开始运行，进入CPU，移出就绪队列")
    proc.status = '运行'


def check_arriving_processes():
    for p in list(filter(lambda x: x.arraiveTime == TIME, process)):
        if p not in readyQueue and p not in ioQueue:
            p.status = '就绪'
            readyQueue.append(p)
            print("进程 ", p.name, " 到达，变为就绪态")
            showReadyQueue(readyQueue)


def check_io_request(proc):
    for r in proc.runInfo:
        if r.timeNode == proc.serveTime - proc.serveTimeLeft:
            if r.operation == '读写磁盘':
                # 处理 I/O 请求
                # 遍历proc.fc，找到当前地址对应的函数名
                for f in proc.fc:
                    if f.rtAddr > proc.currentAddr:
                        break
                    currentFunctionName = f.name
                proc.status = '等待'
                print("占用cpu的进程：", proc.name, "\t函数名：", currentFunctionName,
                      "\t操作类型：", r.operation, "\tI/O操作时间：", r.ioTime)
                showProcess()
                proc.iostart = TIME
                proc.iofinish = TIME + r.ioTime
                ioQueue.append(proc)
                return True
    return False


def calculate_averages():
    global avgTurnAroundTime, avgWeightTurnAroundTime
    avgTurnAroundTime = sum(p.turnAroundTime for p in process if p.status == '完成') /
len(process)
```

```
    avgWeightTurnAroundTime = sum(p.weightTunAroundTime for p in process if p.status == '完成') / len(process)
    print("\n平均周转时间：", avgTurnAroundTime, "\t平均带权周转时间：", avgWeightTurnAroundTime)


def rr():
    global process, TIME, ioQueue, readyQueue, avgTurnAroundTime, avgWeightTurnAroundTime, timeSlice

    create()  # 初始化进程
    print("初始化进程完成")
    showProcess()

    # 初始化时间和队列
    TIME = 0
    print("")
    print("* ", "当前时间：", TIME, " *")
    time.sleep(0.1)
    ioQueue = []
    readyQueue = [p for p in process if p.arraiveTime <= TIME]  # 将到达时间小于等于当前时间的进程加入就绪队列
    print("进程 ", readyQueue[0].name, " 到达，变为就绪态")
    showReadyQueue(readyQueue)

    while readyQueue or ioQueue:
        # 处理就绪队列
        if readyQueue:
            current_process = readyQueue.pop(0)
            print(f"进程 {current_process.name} 开始运行，进入CPU，移出就绪队列")
            showReadyQueue(readyQueue)
            current_process.status = '运行'
            time_spent = 0

            # 执行时间片
            while time_spent < timeSlice and current_process.serveTimeLeft > 0:
                TIME += 1
                time_spent += 1
                current_process.serveTimeLeft -= 1
                print("")
                print("* ", "当前时间：", TIME, " *")
                time.sleep(0.1)

                # 将新到达的进程添加到就绪队列
                for p in list(filter(lambda x: x.arraiveTime == TIME, process)):
                    if p not in readyQueue and p not in ioQueue:
                        p.status = '就绪'
                        readyQueue.append(p)
                        print("进程 ", p.name, " 到达，变为就绪态")
                        showReadyQueue(readyQueue)

                # 检查并处理I/O请求
                for r in current_process.runInfo:
                    if r.timeNode == current_process.serveTime - current_process.serveTimeLeft:
                        # 处理跳转和I/O请求
```

```
                        if r.operation == '跳转':
                            requirePage(r.operateAddr, process.index(current_process))
                            showProcess()
                            # 遍历current_process.fc，找到当前地址对应的函数名
                            for f in current_process.fc:
                                if f.rtAddr > current_process.currentAddr:
                                    break
                                currentFunctionName = f.name
                            # 遍历current_process.fc，找到目标地址对应的函数名
                            for f in current_process.fc:
                                if f.rtAddr > r.operateAddr:
                                    break
                                targetFunctionName = f.name
                            current_process.currentAddr = r.operateAddr
                            print("占用cpu的进程：", current_process.name, "\t函数名：",
currentFunctionName,
                                  "\t操作类型：", r.operation, "\t操作地址：",
r.operateAddr, "\t目标函数名：",
                                  targetFunctionName)
                        elif r.operation == '读写磁盘':
                            # 遍历current_process.fc，找到当前地址对应的函数名
                            for f in current_process.fc:
                                if f.rtAddr > current_process.currentAddr:
                                    break
                                currentFunctionName = f.name
                            current_process.status = '等待'
                            print("占用cpu的进程：", current_process.name, "\t函数名：",
currentFunctionName,
                                  "\t操作类型：", r.operation, "\tI/O操作时间：",
r.ioTime)
                            print("进程 ", current_process.name, " 开始 I/O 请求，变为等待
态")
                            showProcess()
                            current_process.iostart = TIME
                            current_process.iofinish = TIME + r.ioTime
                            ioQueue.append(current_process)
                            break
                    if current_process.status == '等待':
                        break

            # 如果进程完成
            if current_process.serveTimeLeft == 0:
                current_process.status = '完成'
                print(f"进程 {current_process.name} 完成")
                current_process.finishTime = TIME
                current_process.turnAroundTime = TIME - current_process.arraiveTime
                current_process.weightTunAroundTime = current_process.turnAroundTime /
current_process.serveTime
                showProcess()
            elif current_process.status != '等待':
                # 如果进程未完成且不需要I/O，放回队列末尾
                current_process.status = '就绪'
                readyQueue.append(current_process)
                print(f"进程 {current_process.name} 时间片内未完成，放回就绪队列末尾")
                showReadyQueue(readyQueue)

```

```
        # 检查IO队列中是否有完成IO的进程
        for p in list(ioQueue):
            if len(readyQueue) == 0:
                while p.iofinish > TIME:
                    TIME += 1
                    print("")
                    print("* ", "当前时间: ", TIME, " *")
                    time.sleep(0.1)
            if p.iofinish <= TIME:
                print(f"进程 {p.name} I/O 完成，变为就绪态，放回就绪队列末尾")
                p.status = '就绪'
                readyQueue.append(p)
                ioQueue.remove(p)
                showReadyQueue(readyQueue)

        # 将新到达的进程添加到就绪队列
        for p in list(filter(lambda x: x.arraiveTime == TIME, process)):
            if p not in readyQueue and p not in ioQueue:
                p.status = '就绪'
                readyQueue.append(p)
                showReadyQueue(readyQueue)

    # 计算平均周转时间和带权周转时间
    calculate_averages()


def writeResult():
    global process, avgTurnAroundTime, avgWeightTurnAroundTime
    # 检查文件是否存在且非空
    file_exists = os.path.isfile('result.txt') and os.path.getsize('result.txt') > 0

    with open('result.txt', 'a', encoding='utf-8') as file:
        # 如果文件已存在且非空，则在新内容前添加换行符
        if file_exists:
            file.write("\n")
            file.write("************************************************************\n")
            file.write("\n")

        # 假设每个中文字符占两个英文字符宽度
        header_format = "{:<6} {:<7} {:<7} {:<7} {:<7} {:<9}\n"
        data_format = "{:<7} {:<11} {:<11} {:<11} {:<11} {:<15}\n"

        header = header_format.format("进程名", "运行时间", "开始时间", "完成时间", "周转时间", "带权周转时间")
        file.write(header)

        for p in process:
            line = data_format.format(
                p.name, p.serveTime, p.arraiveTime, p.finishTime, p.turnAroundTime, p.weightTunAroundTime)
            file.write(line)

        file.write("\n平均周转时间: {:.5f}\n".format(avgTurnAroundTime))
        file.write("平均带权周转时间: {:.5f}\n".format(avgWeightTurnAroundTime))
```

```python
def init():
    global cpuflag, timeSlice, memoryflag, pageLength, pageFrameAmount
    print("")
    print("**** 选择进程调度算法 ****")
    print("1: FCFS  先来先服务")
    print("2: RR    时间片轮转")
    cpuflag = input("请输入你的选择: ")

    if cpuflag == '2':
        print("")
        timeSlice = int(input("请输入时间片长度(单位ms): "))

    print("")
    print("**** 选择页面调度算法 ****")
    print("1: FIFO  先进先出")
    print("2: LRU   最近最少使用")
    memoryflag = input("请输入你的选择: ")

    print("")
    pageLength = input("请输入页面大小(单位KB): ")
    pageLength = math.ceil(float(pageLength) * 1024)  # 将 KB 转换为 Byte, 对不满 1B 的部分使用 math.ceil() 向上取整

    print("")
    pageFrameAmount = list(map(int, input("请输入为五个进程各自分配的页面数（用空格分开）: ").split()))

    # 确保输入的页面数与进程数匹配
    while len(pageFrameAmount) != 5:
        print("输入的页面数与进程数不匹配, 请重新输入。")
        pageFrameAmount = list(map(int, input("请输入五个进程各自分配的页面数（用空格分开）: ").split()))

    if cpuflag == '1':
        fcfs()
        writeResult()
    elif cpuflag == '2':
        rr()
        writeResult()


def main():
    global mainflag
    print("")
    print("")
    print("*****************************************************************")
    print("*                  CPU & 内存调度管理原型系统                   *")
    print("*    CPU & Memory Scheduling Management Prototype System     *")
    print("*****************************************************************")
    print("*                         **** Menu ****                        *")
    print("*                          1: 设置参数                          *")
    print("*                          0: 退出                              *")
    print("*****************************************************************")
    mainflag = input("请输入你的选择: ")

    if mainflag == '1':
```

```
        init()
        main()
    elif mainflag == '0':
        if platform.system() == 'Windows':
            os.system("cls")  # 如果是 Windows 系统，使用 cls 清屏
        else:
            os.system("clear")  # 如果是 Linux/macOS 系统，使用 clear 清屏
        print("感谢使用！")
        print("")
        if platform.system() == 'Windows':
            # 如果是 Windows 系统，使用 type 命令显示文件内容
            # Windows 默认终端不支持 UTF-8 编码，如遇乱码请查找相关资料或换用其它终端解决
            os.system("type result.txt")
            print("")
        else:
            # 如果是 Linux/macOS 系统，使用 cat 命令显示文件内容
            os.system("cat result.txt")
            print("")


if __name__ == '__main__':
    main()

```