学生学号	0122210880414	实验课成绩	

武汉理工大学

学 生 实 验 报 告 书

实验课程名称 操作系统

开 课 学 院 计算机与人工智能学院

指导老师姓名 陆丽萍

学 生 姓 名 周豪

学生专业班级 软件 zy2201

2024 — 2025 学年 第 1 学期

实验课程名称：操作系统

实验项目名称	动态分区管理			实验成绩	
实验者	周豪	专业班级	软件 zy2201	组别	
同组者	无			实验日期	2024 年 12 月 11 日

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

1．掌握分区管理的基础思想，加深对存储管理的理解；

2．掌握动态分区的分配，分析各种分配算法的特点及区别；

3．掌握动态分区的回收，分析碎片问题并掌握紧凑技术。

二、实验基本原理与设计（包括实验方案设计，实验手段的确定，试验步骤等，用硬件逻辑或者算法描述）

学习动态分区的管理思想，选择 1~3 种内存分配算法（最先适应法、最佳适应法、最坏适应法）模拟实现动态分区的内存分配回收：

1.能够输入给定的内存大小，进程的个数，每个进程所需内存空间的大小等；

2.能够选择分配或回收操作，并能显示完成内存分配或回收后内存空间的使用情况；

3.能够显示进程在内存的存储地址、大小等。

三、主要仪器设备及耗材

Window 10
Python 3.9
PyQt5

第二部分：实验调试与结果分析（可加页）

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

（1）数据结构

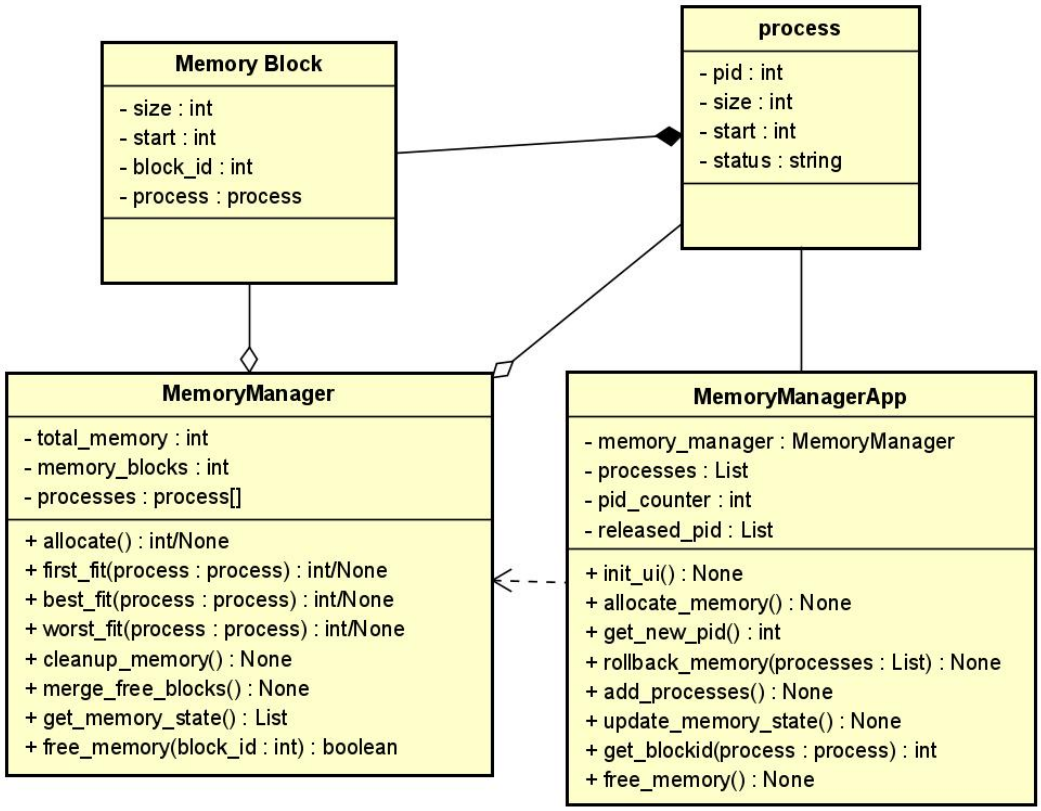

图 1 类图

在本次实验实现过程中，我创建了四个主要的类：MemoryManagerApp、MemoryManager、Process 和 MemoryBlock，共同构成了内存管理系统的核心功能模块。

MemoryManagerApp 类是应用程序的主要入口，负责初始化用户界面、处理用户输入以及与内存管理系统的交互。它通过一个 MemoryManager 实例来管理内存的分配和回收，并且负责跟踪当前的进程和内存状态。

MemoryManager 类负责具体的内存管理工作，提供内存分配和回收的功能，包含一个 memory_blocks 属性，用于存储所有内存块，并通过不同的内存分配策略来执行内存分配操作。每当内存被分配或释放时，MemoryManager 会更新内存状态，并提供方法来查询当前的内存使用情况。

Process 类代表一个正在运行的进程，它有一个唯一的 pid，以及需要分配的内存大小。Process 类还维护着与之关联的内存块，当一个进程结束时，相关的内存块将会被释放回内存池。

MemoryBlock 类表示一个内存块，它具有 block_id，start，size 等属性。每个 MemoryBlock 实例要么处于空闲状态，要么已经被某个 Process 占用，内存块的状态会随进程的内存需求变化而更新。

（2）三种分配方法

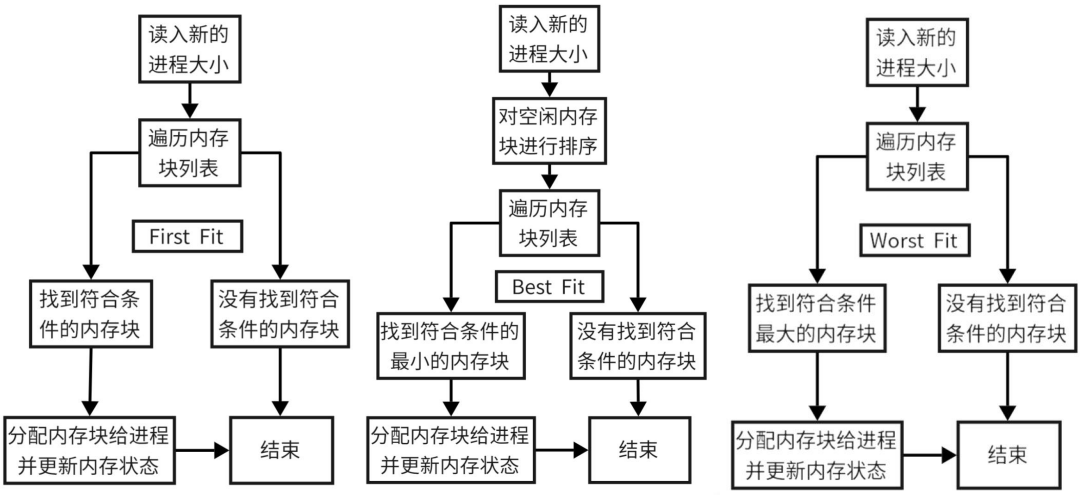

图 2 三种分配方法

最先适应法从内存的开始位置依次遍历内存块，找到第一个满足进程内存需求的空闲内存块，并将其分配给进程。如果找到合适的内存块，则立即分配；如果没有合适的内存块，分配失败。

最佳适应法首先对所有的空闲内存块按照大小进行排序，选择最小的一个且能够满足进程需求的内存块进行分配。

最坏适应法选择最大的空闲内存块进行分配，将进程放置在剩余空间最大的内存块中。

（3）主要函数实现

allocate_memory()

用于处理内存分配的逻辑。用户输入总内存大小、进程数量及各个进程的内存需求后，该函数会检查是否有足够的内存可供分配。如果内存足够，它会创建进程对象并使用选定的分配策略进行内存分配。如果分配成功，内存状态将更新；如果分配失败，所有操作将回滚。

add_processes()

用于添加新的进程。当用户输入新的进程内存需求时，系统会首先检查当前剩余的空闲内存是否足够。若足够，则创建新进程并使用选定的分配策略进行内存分配。如果内存不足或分配失败，将回滚新增进程的分配，并打印错误信息。

rollback_memory()

用于回滚分配失败的进程。当内存分配操作失败时，rollback_memory()会释放已经分配的内存，并回收被分配给进程的内存块。同时，它还会将已回收的进程编号放入一个回收列表，以便下次分配时可以重用这些编号。

update_memory_state()

用于更新内存的状态，并将当前的内存状态显示在用户界面中。它会遍历内存管理器的内存块，检查每个内存块是否被分配给某个进程。如果是，它会更新表格中显示的进程编号、内存起始地址和大小。如果是空闲内存块，则显示空闲状态。

get_block_id_by_process()

通过进程的起始地址来查找对应的内存块 ID。它遍历内存管理器中的所有内存块，寻找与进程对应的内存块，并返回该内存块的 ID，以便进行后续的内存释放操作。

free_memory()

用于回收用户指定进程的内存。用户输入进程编号后，系统会查找对应的进程，并释放其所占用的内存块。如果该进程存在并成功释放内存，内存状态将更新，并删除进程记录；如果输入无效，则提示错误信息。

二．实验结果及分析（包括结果描述、实验现象分析、影响因素讨论、综合分析和结论等）

图 3 界面原型与成果展示

本系统的 UI 界面是采用 PyQt5 完成的，包含三个主要部分：内存与进程配置区、操作区和内存状态表格。在配置区，用户可以输入内存总大小、进程数量以及每个进程的内存需求，并通过“分配内存”按钮启动内存分配操作。操作区提供了回收内存、新增进程和选择内存分配策略（包括最先适应法、最佳适应法、最坏适应法）的功能，用户可通过输入框和按钮进行操作。内存状态表格实时展示当前内存分配情况，显示每个内存块的编号、起始地址、大小和分配状态，帮助用户清晰了解内存的使用情况。

三．实验小结、建议及体会

本次实验是基于课程中的内存分配内容展开的。我实现了一个动态内存分配模拟系统，采用了最先适应法、最佳适应法和最坏适应法来模拟内存的分配与回收。在这个过程中，我更好地理解了内存管理算法的工作原理，并把它们应用到实际的内存分配操作中。

首先，我用 PyQt5 设计了一个简单易用的界面，用户可以输入总内存、进程数和每个进程所需的内存空间等参数来配置系统。每种分配方法都有对应的按钮，用户可以根据自己的需求选择不同的分配策略。系统也允许用户回收内存，进行内存的动态管理。每次分配或回收内存后，内存的状态会实时显示在表格中，用户可以看到每个内存块的状态、进程在内存中的起始地址和大小等信息。

通过这次实验，我深刻体会到内存管理的重要性和复杂性，让我更好地理解了操作系统中内存分配算法的实现，也加深了对内存管理基本原理的理解。

实验课程名称：操作系统

实验项目名称	磁盘调度			实验成绩	
实验者	周豪	专业班级	软件 zy2201	组别	
同组者	无			实验日期	2024 年 12 月 13 日

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

1．掌握文件的物理结构，分析不同物理结构的区别；

2．掌握文件存储设备的特性，分析顺序、直接存储设备的特点及区别；

3．掌握常用的磁盘调度算法（先来先服务法、最短寻道时间优先、电梯算法）；

4. 掌握磁盘调度性能评价的准则，并能对调度性能进行定量评价。

二、实验基本原理与设计（包括实验方案设计，实验手段的确定，试验步骤等，用硬件逻辑或者算法描述）

磁盘调度是文件存储管理的核心内容。按照下列要求用高级语言编写和调试一个磁盘调度程序，以便加深了解有关磁盘等直接存储设备的概念，并体会磁盘调度算法的具体实施过程。

选择 1~3 种磁盘调度算法（先来先服务法、最短寻道时间优先、电梯算法）模拟实现磁盘调度：

要求：

1.能够输入当前磁头的位置、磁头移动方向、磁道访问请求序列等；

2.计算磁头移动的总磁道数；

3.能够显示磁盘调度结果（磁头依次访问的磁道号顺序等）。

三、主要仪器设备及耗材

Window 10
Python 3.9
PyQt5

第二部分：实验调试与结果分析（可加页）

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

（1）数据结构

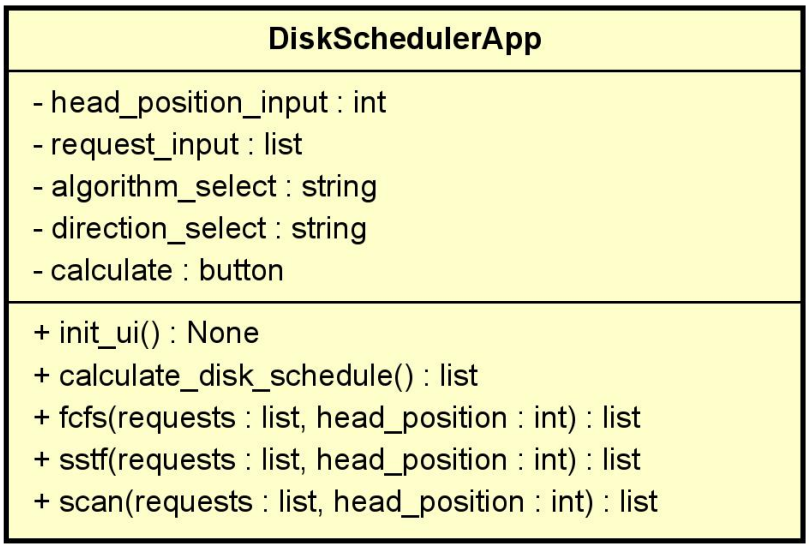

图 1 DiskSchedulerApp 类图

DiskSchedulerApp 是一个基于 PyQt5 的启动类，其包含一个文本框 head_position_input 处理磁头位置的输入，request_input 处理对于查询队列的输入，algorithm_select 则可以通过下拉框进行三种不同方法的选择，direction_select 则只针对电梯算法，选择从小到大或是从大到小进行扫描，同样采用下拉框输入，最后是 calculate 按钮，点击即可计算结果，展示最短寻道长度及路径。

（2）算法实现

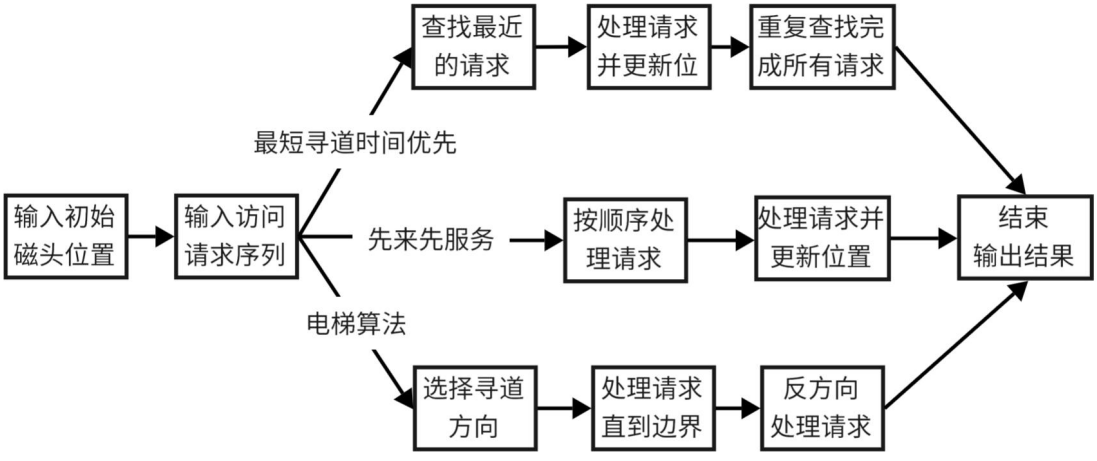

图 2 三种寻道算法过程

A.先来先服务法

在实现先来先服务法（FCFS）时，我首先获取磁头的初始位置和所有待处理的请求。我按照请求的到达顺序处理这些请求。每次，我计算当前磁头位置到目标请求的距离，并更新磁头的位置。所有请求处理完后，我就能够计算出磁头的总移动距离。这个算法的实现比较简单，因为它不对请求的顺序进行优化，只是按照到达的顺序依次处理每个请求。

B.最短寻道时间优先 (SSTF)

在实现最短寻道时间优先（SSTF）时，我首先对请求进行排序。然后，从当前磁头的位置出发，我会选择与当前磁头位置距离最近的请求进行处理。每次处理完一个请求，我就更新磁头位置，并重新计算剩余请求与磁头的距离，选择下一个最短距离的请求进行处理，直到所有请求都被处理完。这个算法的关键是每次选择最短寻道时间的请求，从而尽量减少磁头的移动距离。

C.电梯算法 (SCAN)

实现电梯算法（SCAN）时，我首先对请求进行排序，然后根据磁头的初始位置和扫描方向（从小到大或从大到小），决定磁头的移动方向。磁头会朝着指定方向移动，直到到达磁盘的边缘或者处理完所有在该方向上的请求。当磁头到达边界时，我会反转方向，继续处理另一个方向上的请求。整个过程不断更新磁头的位置，直到所有请求都处理完。电梯算法的关键是在每次移动时，确保按照指定的方向遍历所有请求，并计算出总的寻道长度。

二、实验结果及分析（包括结果描述、实验现象分析、影响因素讨论、综合分析和结论等）

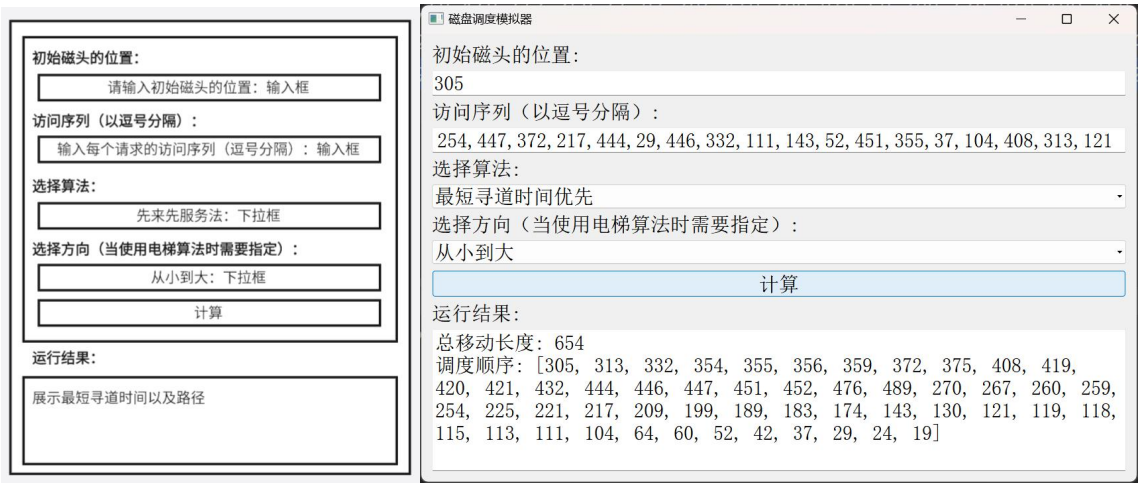

图 3 界面原型及成果展示

与实验 1 一样，本次实验的 UI 界面我也使用的 PyQt 进行的实现。首先，用户需要在初始磁头位置框里输入磁头的起始位置，然后在访问序列框中输入需要访问的磁道号，多个磁道号用逗号隔开。接下来，我可以从下拉菜单中选择一种调度算法（比如先来先服务法、最短寻道时间优先法或电梯算法），如果选择的是电梯算法，还需要指定磁头的扫描方向。点击“计算”按钮后，程序会自动计算出磁头的总移动长度和访问的顺序，然后把结果显示在下方的运行结果区域。

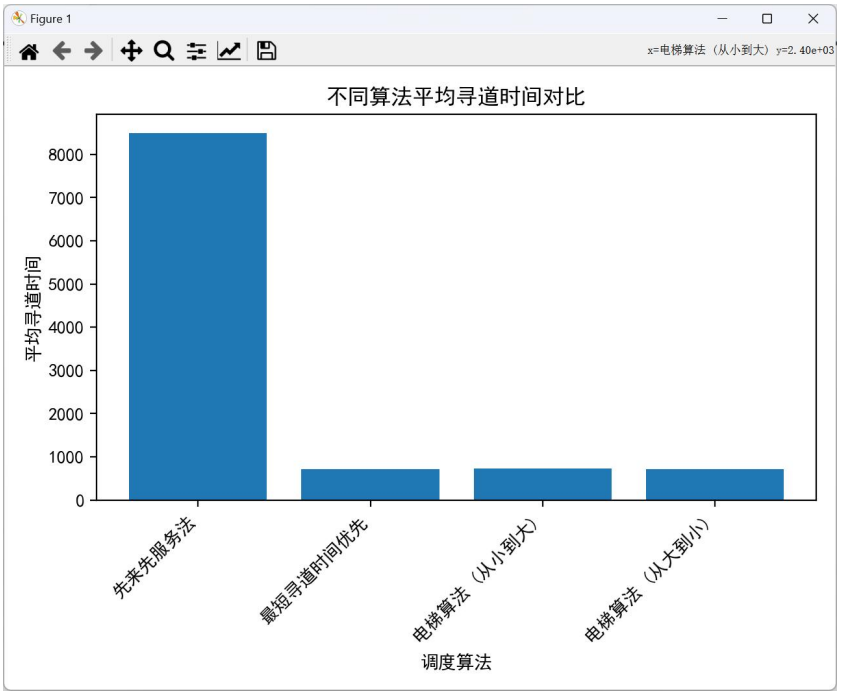

图 4 三种寻道算法比较

出于对三种算法效率的兴趣，我额外设计了对三种算法寻道平均时间的比较。首先我编写了 py 脚本，用于随机生成大量测试用例。对于本次实验，我生成了 100 组测试用例，每组包含一个初始磁头位置和 50 个请求，每个请求的位置在（0，500）之间。将生成的大测试样例存入.txt 文件。然后我编写了测试代码，从上述.txt 文件中依次读取每条测试用例，然后使用三种算法进行求解并求出寻道时间平均值，利用 matplotlib 绘制可视化柱状图，如图 3 所示。具体结果为：

1.先来先服务法: 8496.25

2.最短寻道时间优先: 714.84

3.电梯算法 (从小到大): 725.28

4.电梯算法 (从大到小): 720.64

可以看出先来先服务算法的效率最低，达到最短寻道时间优先算法的 11.9 倍，以及电梯算法的 11.8 倍。而最短寻道时间优先算法与电梯算法的效率则非常接近。

三、实验小结、建议及体会

通过本次磁盘调度模拟实验，我实现了三种磁盘调度算法：先来先服务法（FCFS）、最短寻道时间优先（SSTF）和电梯算法（SCAN）。在实现过程中，我深入理解了每种算法的原理及其特点，分别通过处理请求到达顺序、选择最短寻道时间的请求以及在两个方向上扫描处理请求来优化磁头的移动。在 UI 展示方面，我使用 PyQt5 开发了一个直观的界面，方便用户输入参数并查看结果。通过对三种算法的测试与比较，发现先来先服务法的效率最低，而最短寻道时间优先和电梯算法的效率较高且接近。此次实验帮助我加深了对磁盘调度算法的理解。