

江蘇大學京江學院

JIANGSU UNIVERSITY JINGJIANG COLLEGE



操作系统实验报告

题 目：	实验四 驱动调度
上机时间：	2023.12.07
授课教师：	潘雨青
姓 名：	马云骥
学 号：	4211153047
专 业：	软件工程
班 级：	J软件(嵌入)(专转本)2102
日 期：	2023.12.10

实验四 驱动调度

1 实验目的

1. 熟悉典型的驱动调度算法
2. 能够仿真实验典型的驱动调度算法

2 实验设计

原始参考的案例代码均由  C# 编写，这里本人使用  Python 实现，力求达到同样的效果。

开发环境:  Windows 11 Pro 23H2

 Python 3.12

 PyCharm 2023.3

实验指导书实验内容

1. 运行实例中的 DriverScheduling 项目，了解其程序结构，并理解其中的驱动调度算法是采用了何种驱动调度算法，并在实验报告中回答。
2. 在实例中的 DriverScheduling 项目实现按电梯调度算法。

先回答指导书中的问题，原实例模拟了 SSTF（最短寻道时间优先）的磁盘驱动调度算法。SSTF 算法的工作原理是选择磁盘请求队列中距离当前磁头位置最近的请求进行处理，这种方法减少了磁头的平均寻道时间，但可能导致饥饿问题。

下面是我的 Python 实现：

```
1 def sstf(requests, start):
2     # 初始化变量
3     size = len(requests) # 请求的数量
4     sequence = [] # 请求处理顺序
5     distance = 0 # 总移动距离
6     current_position = start # 当前磁头位置
7     movements = [] # 移动记录
8     last_direction = None # 上一次移动的方向
9
10    # 当所有请求都被处理时停止
11    while len(sequence) < size:
12        # 找到最近的请求
13        closest_request = min(requests, key=lambda x: abs(x - current_position))
14        # 确定移动方向
15        direction = "up" if closest_request > current_position else "down"
16        # 累计移动距离
17        distance += abs(current_position - closest_request)
18        # 记录移动
```

```

19         if last_direction is not None and direction != last_direction:
20             movements.append(f"From {current_position} turn around to {closest_request}
and trip is {distance}")
21         else:
22             movements.append(f"From {current_position} move to {closest_request} and trip
is {distance}")
23             # 更新位置和方向
24             current_position = closest_request
25             sequence.append(closest_request)
26             requests.remove(closest_request)
27             last_direction = direction
28
29         return sequence, distance, movements
30
31
32 def scan(requests, start, direction=None):
33     # 确定初始扫描方向
34     if direction is None:
35         nearest_up = min((req for req in requests if req > start), default=float('inf'))
36         nearest_down = max((req for req in requests if req < start), default=float('-
inf'))
37         direction = 'up' if abs(nearest_up - start) < abs(nearest_down - start) else
'down'
38
39     requests.sort()
40     sequence = []
41     distance = 0
42     current_position = start
43     movements = []
44
45     if direction == 'up':
46         # 处理向上的请求
47         up_requests = [r for r in requests if r >= current_position]
48         for r in up_requests:
49             distance += abs(current_position - r)
50             movements.append(f"From {current_position} move to {r} and trip is
{distance}")
51             current_position = r
52             sequence.append(r)
53         # 改变方向
54         if up_requests:
55             next_request_down = max((req for req in requests if req < start),
default=current_position)
56             if next_request_down != current_position:
57                 distance += abs(current_position - next_request_down)
58                 movements.append(f"From {current_position} turn around to
{next_request_down} and trip is {distance}")
59                 current_position = next_request_down
60                 sequence.append(next_request_down)
61             for r in reversed(requests):
62                 if r < start and r not in sequence:
63                     distance += abs(current_position - r)
64                     movements.append(f"From {current_position} move to {r} and trip is
{distance}")
65                     current_position = r
66                     sequence.append(r)

```

```

67     else:
68         # 处理向下的请求
69         down_requests = [r for r in requests if r <= current_position]
70         for r in reversed(down_requests):
71             distance += abs(current_position - r)
72             movements.append(f"From {current_position} move to {r} and trip is
{distance}")
73             current_position = r
74             sequence.append(r)
75         # 改变方向
76         if down_requests:
77             next_request_up = min((req for req in requests if req > start),
default=current_position)
78             if next_request_up != current_position:
79                 distance += abs(current_position - next_request_up)
80                 movements.append(f"From {current_position} turn around to
{next_request_up} and trip is {distance}")
81                 current_position = next_request_up
82                 sequence.append(next_request_up)
83             for r in requests:
84                 if r > start and r not in sequence:
85                     distance += abs(current_position - r)
86                     movements.append(f"From {current_position} move to {r} and trip is
{distance}")
87                     current_position = r
88                     sequence.append(r)
89
90         return sequence, distance, movements
91
92
93 def main():
94     # 用户输入选择算法
95     algorithm_choice = input("请选择要模拟的算法 (1.SSTF, 2.SCAN): ").strip()
96
97     # 输入磁头起始位置
98     start_input = input("输入磁头起始位置 (默认 15): ").strip()
99     start = int(start_input) if start_input else 15
100
101     # 输入磁盘请求序列
102     requests_input = input("输入磁盘请求序列 (使用半角逗号分隔, 默认
[2,34,5,8,3,24,12,18]): ").strip()
103     requests = [int(r) for r in requests_input.split(',') if requests_input else [2, 34,
5, 8, 3, 24, 12, 18]
104
105     # 根据用户选择运行相应算法
106     if algorithm_choice == '1':
107         sequence, distance, movements = sstf(requests.copy(), start)
108     elif algorithm_choice == '2':
109         direction_input = input("输入起始扫描方向 (1.up, 2.down, 默认为最近的被请求磁道的方
向): ").strip()
110         direction = None
111         if direction_input == '1':
112             direction = 'up'
113         elif direction_input == '2':
114             direction = 'down'
115         sequence, distance, movements = scan(requests.copy(), start, direction)

```

```

116         else:
117             print("无效的选择。")
118             return
119
120         # 打印算法执行结果
121         print("Start...")
122         for movement in movements:
123             print(movement)
124         print("-----Finish")
125
126
127 if __name__ == "__main__":
128     main()
129

```

我将本次实验需要模拟的两种磁盘驱动调度算法实现在了一个 Python 程序中，程序运行后可以根据需要，选择使用对应的算法进行模拟。

2.1 SSTF (`sstf` 函数)

- **目的：**模拟最短寻道时间优先算法，选择距当前磁头位置最近的请求，以最小化磁头移动距离。
- **参数：**
 - `requests`：磁盘请求的列表。
 - `start`：磁头的起始位置。
- **过程：**
 - 循环处理所有请求，每次选择最近的请求。
 - 计算磁头移动距离并记录移动方向。
 - 更新当前磁头位置和处理顺序。

2.2 SCAN (`scan` 函数)

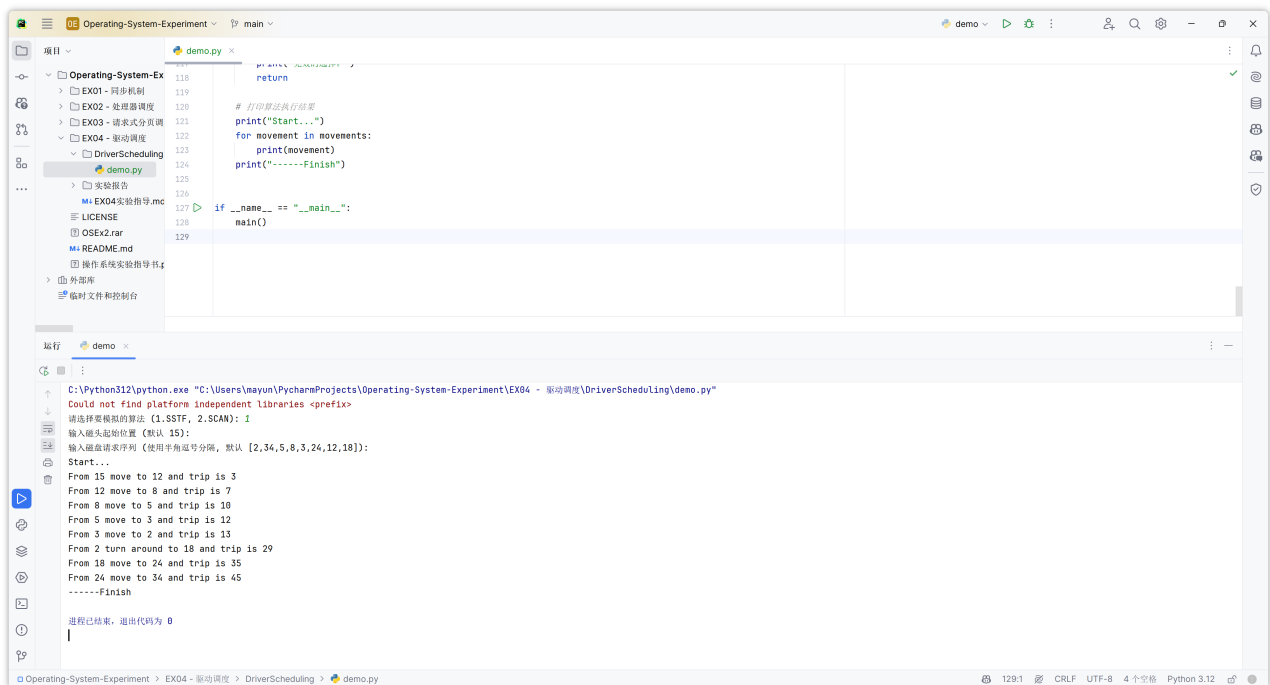
- **目的：**模拟电梯算法（扫描算法），按一个方向移动磁头，处理所有请求，直到到达最远的请求，然后改变方向。
- **参数：**
 - `requests`：磁盘请求的列表。
 - `start`：磁头的起始位置。
 - `direction`：起始扫描方向（可选）。
- **过程：**
 - 确定初始扫描方向（如果未指定）。
 - 处理当前方向上的请求。
 - 到达最远端后改变方向，并处理剩余请求。

2.3 主函数（`main` 函数）

- 流程：
 - 让用户选择算法（SSTF或SCAN）。
 - 获取磁头起始位置和磁盘请求序列。
 - 根据用户选择执行相应算法。
 - 打印算法执行的详细移动过程和结果。

3 实验结果

3.1 SSTF（最短寻道时间优先）



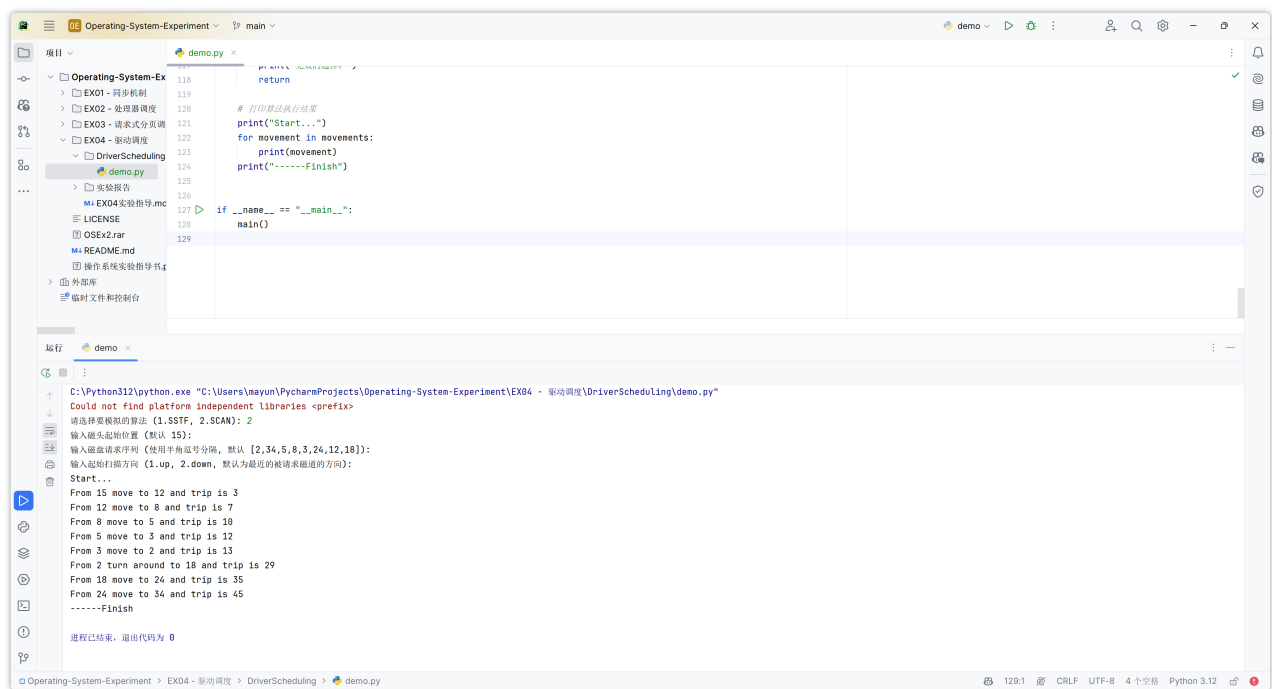
```
demo.py
118     return
119
120     # 打印算法执行结果
121     print("Start...")
122     for movement in movements:
123         print(movement)
124     print("-----Finish")
125
126
127 if __name__ == "__main__":
128     main()
129
```

运行 demo

```
C:\Python312\python.exe "C:\Users\mayun\PycharmProjects\Operating-System-Experiment\EX04 - 驱动调度\DriverScheduling\demo.py"
Could not find platform independent libraries <prefix>
请选择要模拟的算法 (1. SSTF, 2. SCAN): 1
输入磁头起始位置 (默认 15):
输入磁盘请求序列 (使用半角逗号分隔, 默认 [2,34,5,8,3,24,12,18]):
Start...
From 15 move to 12 and trip is 3
From 12 move to 8 and trip is 7
From 8 move to 5 and trip is 10
From 5 move to 3 and trip is 12
From 3 move to 2 and trip is 13
From 2 turn around to 18 and trip is 29
From 18 move to 24 and trip is 35
From 24 move to 34 and trip is 45
-----Finish

进程已结束, 退出代码为 0
```

3.2 SCAN（电梯算法）



4 实验总结

本实验通过模拟两种常用的磁盘调度算法——最短寻道时间优先（SSTF）和电梯算法（SCAN），成功地展示了它们在处理磁盘请求时的行为模式和效率差异。

- **SSTF算法**：在该算法中，磁头总是移动到最近的请求位置。这种方法可以有效减少磁头移动距离，从而提高处理速度。然而，它也可能导致远离当前磁头位置的请求长时间得不到处理，造成“饥饿”现象。
- **SCAN算法**：通过模拟电梯运行方式的 SCAN 算法，更加公平地处理每个请求。它通过在一个方向上连续处理请求，直到到达最远端，然后改变方向继续处理其余请求。这种方式虽然可能增加磁头的总移动距离，但它避免了 SSTF 算法中的“饥饿”问题，保证了所有请求最终都会得到处理。

总体来看，这两种算法各有优劣。SSTF 算法适用于请求集中且较为均匀分布的情况，能有效提高处理速度。而 SCAN 算法则适用于请求分布较为广泛，且对响应时间公平性有较高要求的场景。通过本实验，我们不仅加深了对这些经典磁盘调度算法的理解，还为未来在实际系统中选择合适的调度策略提供了参考依据。