江苏大学京江学院
JIANGSU UNIVERSITY JINGJIANG COLLEGE


# 操作系统实验报告

| | |
|---|---|
| 题　　目： | 实验二　处理器调度 |
| 上机时间： | 2023.11.23 |
| 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： | 2023.11.30 |

# 实验二　处理器调度

## 1　实验目的

1. 熟悉典型的处理器调度算法
2. 熟悉在编程中使用时钟定时器
3. 能够仿真实现典型的处理器调度算法

## 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 2023.2.5

> 实验指导书实验内容
>
> 1. 运行实例中的 ProcessScheduling 项目，了解其程序结构，并理解其中的处理器调度方法是采用了何种调度算法，并在实验报告中回答。
> 2. 在实例中的 ProcessScheduling 项目实现按优先级调度，以及按优先级的时间片调度。

先回答指导书中的问题，原实例模拟了两种处理器调度算法，一种是 FIFO（先进先出），另一种是 FIFO + 时间片轮转的调度算法。

下面是我的 Python 实现：

```python
import random
import time

class PCB:
    def __init__(self, name, run_duration, priority):
        self.name = name  # 进程名称
        self.run_duration = run_duration  # 运行时间
        self.priority = priority  # 优先级
        self.status = "就绪"  # 初始状态设置为"就绪"

    def display(self):
        # 打印进程的状态信息
        print(f"Process {self.name} is {self.status} and prio is {self.priority}, RunDuration is {self.run_duration}")


class ReadyList:
    def __init__(self):
```

```
        self.plist = []  # 准备队列

    def push(self, pcb):
        # 将进程添加到队列中
        self.plist.append(pcb)

    def pop(self):
        # 从队列中取出一个进程
        return self.plist.pop(0) if self.plist else None

    def sort_by_priority(self):
        # 按优先级对队列进行排序
        self.plist.sort(key=lambda pcb: pcb.priority)

    def is_empty(self):
        # 检查队列是否为空
        return len(self.plist) == 0

    def display(self):
        # 打印队列中所有进程的状态
        for pcb in self.plist:
            pcb.display()


def FIFO_scheduling(ready_list):
    # 先进先出调度算法
    while not ready_list.is_empty():
        current_pcb = ready_list.pop()
        while current_pcb.run_duration > 0:
            print("================================")
            current_pcb.status = "运行"
            current_pcb.run_duration -= 1
            current_pcb.display()
            ready_list.display()
            print("================================")
            time.sleep(1)
        current_pcb.status = "完成"
        current_pcb.display()


def FIFO_time_slice(ready_list, time_slice):
    # 带时间片的先进先出调度算法
    while not ready_list.is_empty():
        current_pcb = ready_list.pop()
        for _ in range(time_slice):
            if current_pcb.run_duration == 0:
                break
            print("================================")
            current_pcb.status = "运行"
            current_pcb.run_duration -= 1
            current_pcb.display()
            ready_list.display()
            print("================================")
            time.sleep(1)
        if current_pcb.run_duration > 0:
            current_pcb.status = "就绪"
```

```
            ready_list.push(current_pcb)
        else:
            current_pcb.status = "完成"
            current_pcb.display()


def priority_scheduling(ready_list):
    # 优先级调度算法
    while not ready_list.is_empty():
        ready_list.sort_by_priority()
        current_pcb = ready_list.pop()
        while current_pcb.run_duration > 0:
            print("================================")
            current_pcb.status = "运行"
            current_pcb.run_duration -= 1
            current_pcb.display()
            ready_list.display()
            print("================================")
            time.sleep(1)
        current_pcb.status = "完成"
        current_pcb.display()


def priority_time_slice(ready_list, time_slice):
    # 带时间片的优先级调度算法
    while not ready_list.is_empty():
        ready_list.sort_by_priority()
        current_pcb = ready_list.pop()
        for _ in range(time_slice):
            if current_pcb.run_duration == 0:
                break
            print("================================")
            current_pcb.status = "运行"
            current_pcb.run_duration -= 1
            current_pcb.display()
            ready_list.display()
            print("================================")
            time.sleep(1)
        if current_pcb.run_duration > 0:
            current_pcb.status = "就绪"
            ready_list.push(current_pcb)
        else:
            current_pcb.status = "完成"
            current_pcb.display()


def main():
    algorithms = {
        "1": FIFO_scheduling,
        "2": lambda rl: FIFO_time_slice(rl, 1),
        "3": priority_scheduling,
        "4": lambda rl: priority_time_slice(rl, 1)
    }

    print("选择处理器调度算法")
    print("1: 先进先出")
```

```
    print("2: 先进先出+时间片轮转")
    print("3: 优先级")
    print("4: 优先级+时间片轮转")
    choice = input("请输入你的选择: ")
    ready_list = ReadyList()

    # 生成5个随机PCB
    for i in range(1, 6):
        pcb = PCB(f"PCB{i}", random.randint(3, 6), random.randint(1, 3))  # 运行时间3-6, 优先级1-3
        ready_list.push(pcb)

    print("开始模拟:")
    ready_list.display()
    print("********************************")

    # 运行调度算法
    if choice in algorithms:
        algorithms[choice](ready_list)
    else:
        print("无效的选择! ")


if __name__ == "__main__":
    main()

```

我将本次实验需要模拟的四种处理器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。程序定义了几个类和函数来模拟进程的创建、管理和调度。下面是各部分的详细解释：

## 2.1 类 `PCB`（进程控制块）

- `__init__(self, name, run_duration, priority)`：初始化函数，用于创建一个新的进程控制块（PCB）。它接受进程名称 `name`、运行时间 `run_duration` 和优先级 `priority` 作为参数。
- `display(self)`：用于打印当前进程的状态、优先级和剩余运行时间。

## 2.2 类 `ReadyList`

- `__init__(self)`：初始化函数，创建一个空的就绪队列。
- `push(self, pcb)`：将一个进程（PCB）添加到就绪队列中。
- `pop(self)`：从就绪队列中取出（并删除）第一个进程。
- `sort_by_priority(self)`：根据优先级对就绪队列中的进程进行排序。
- `is_empty(self)`：检查就绪队列是否为空。
- `display(self)`：打印就绪队列中所有进程的信息。

### 2.3 调度算法函数

- `FIFO_scheduling(ready_list)`：实现先进先出（FIFO）调度算法。不断从就绪队列中取出进程并运行，直到队列为空。
- `FIFO_time_slice(ready_list, time_slice)`：实现带时间片的FIFO调度。每个进程运行固定的时间片后，重新回到队列末尾（如果还有剩余运行时间）。
- `priority_scheduling(ready_list)`：实现基于优先级的调度。总是选取优先级最高的进程运行。
- `priority_time_slice(ready_list, time_slice)`：结合优先级和时间片的调度算法。优先级高的进程先运行，每个进程的运行时间受时间片限制。

### 2.4 主函数 `main`

- 提供一个菜单供用户选择不同的调度算法。
- 随机生成5个进程，并将它们添加到就绪队列。
- 根据用户的选择执行相应的调度算法。

### 2.5 运行流程

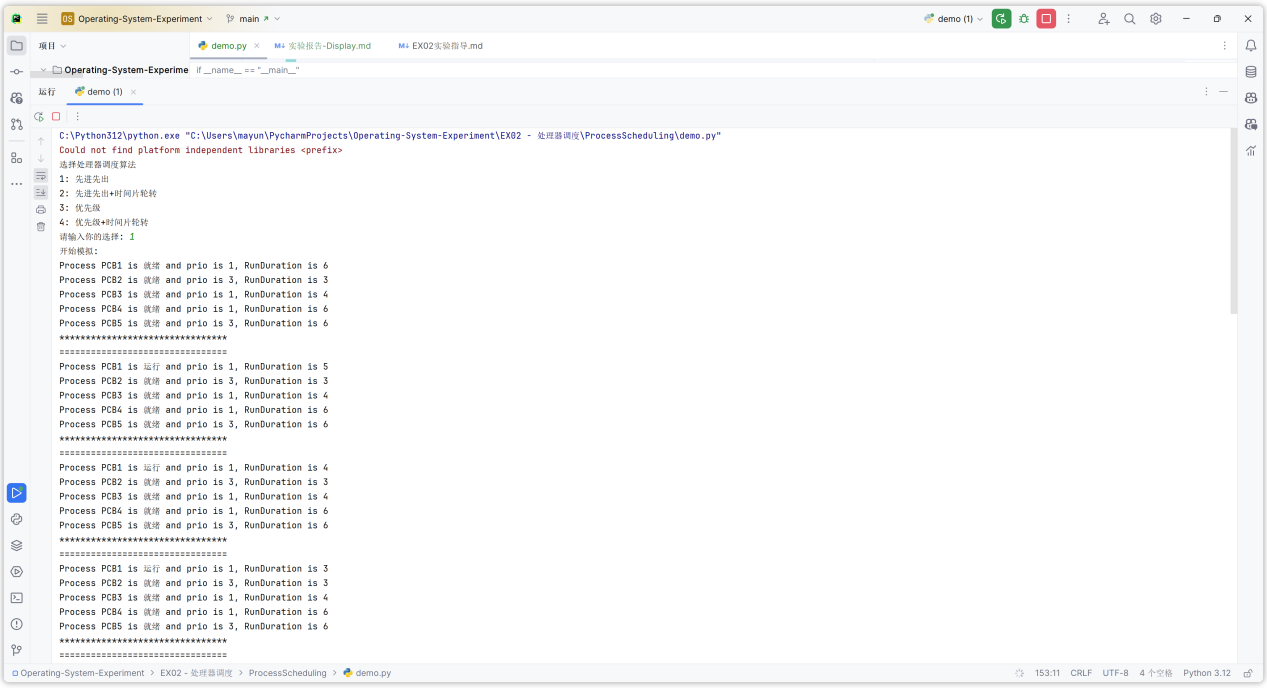
- 程序开始时，用户选择一个调度算法。
- 程序创建5个具有随机运行时间和优先级的进程。
- 根据所选算法，程序模拟进程调度的过程，显示每个进程的状态和就绪队列的变化。

### 2.6 其他

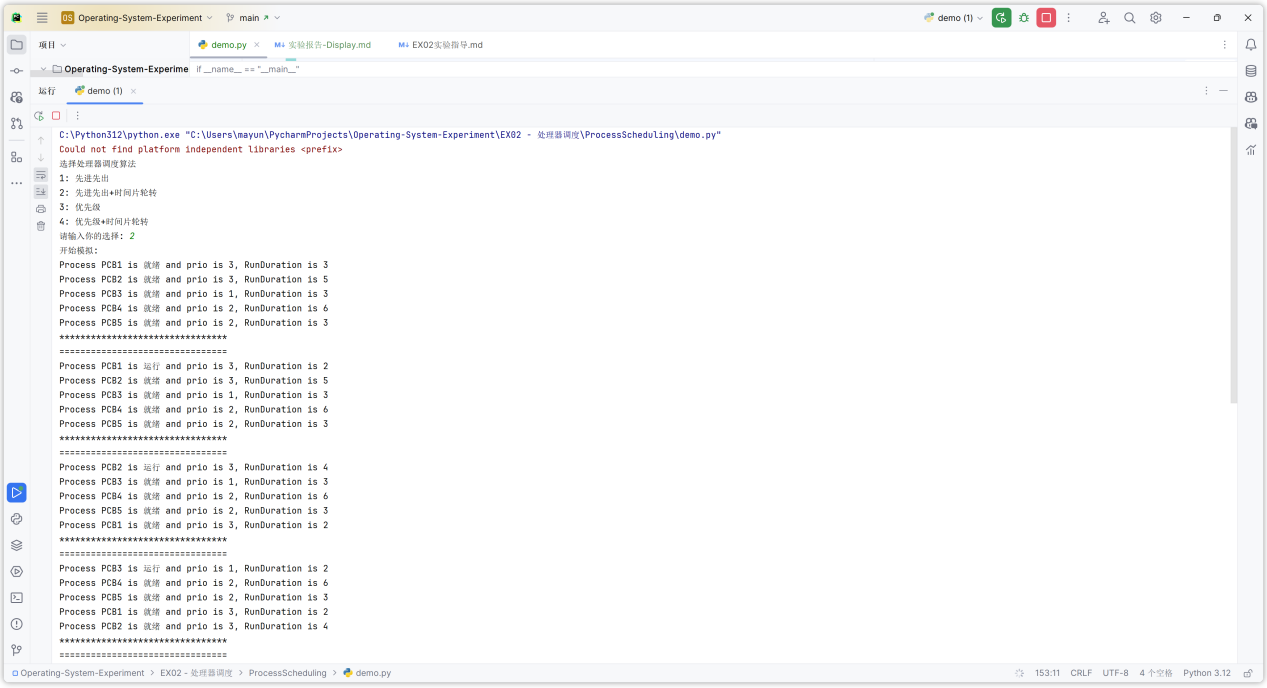
- 程序使用了 `time.sleep(1)` 来模拟进程运行，使得每次状态变更之间有一秒的间隔。
- 这种模拟有助于理解不同调度算法的工作原理和区别。

## 3 实验结果

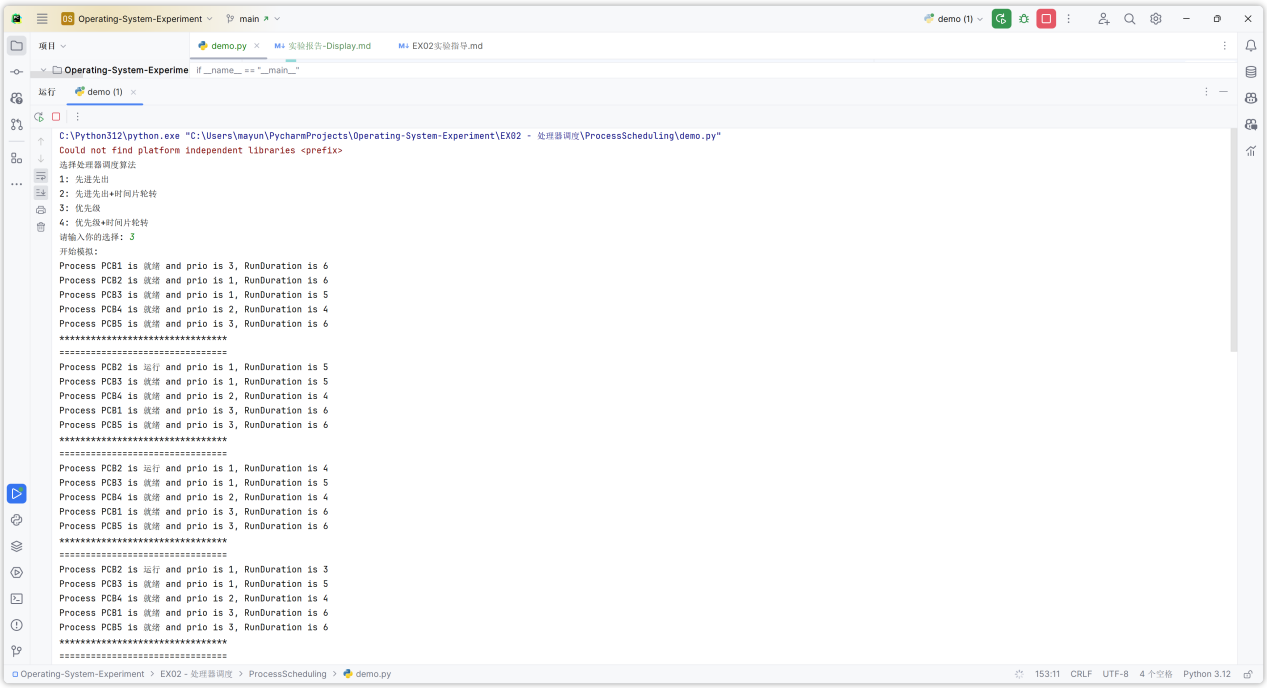
### 3.1 FIFO（先进先出）



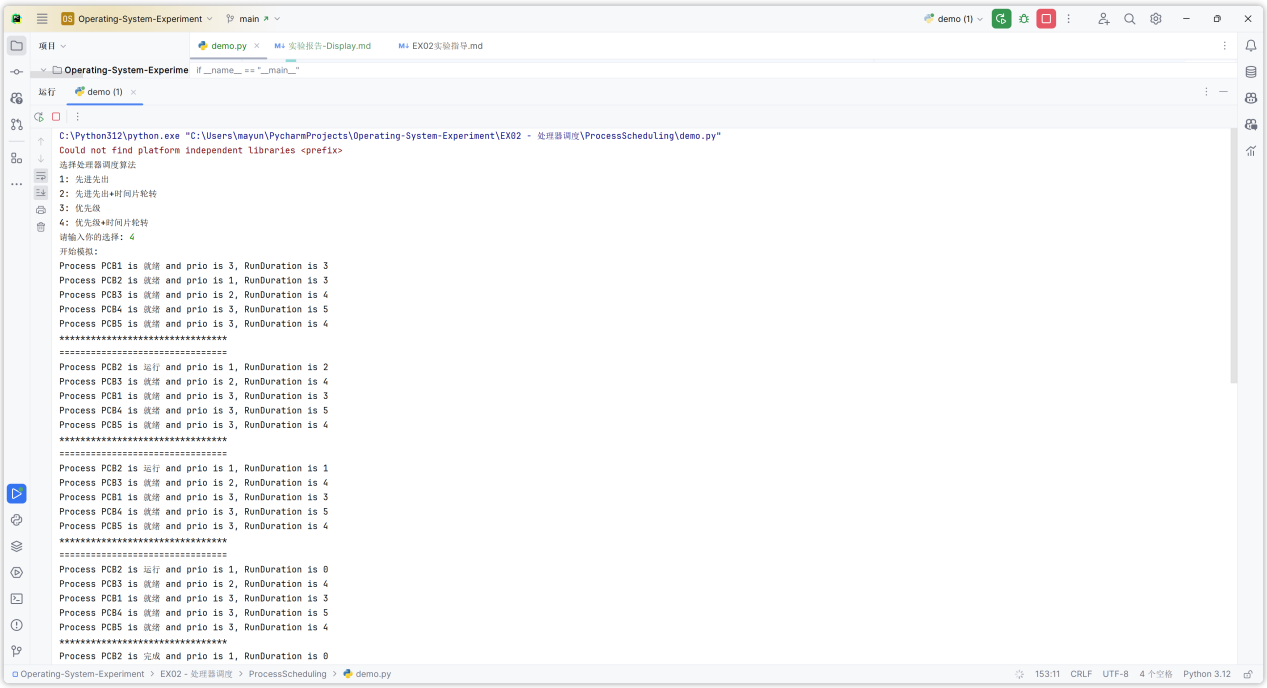

## 3.2 FIFO + 时间片轮转



## 3.3 优先级




## 3.4 优先级 + 时间片轮转




# 4 实验总结

通过这次实验，我不仅加深了对处理器调度算法的理解，也提高了我的编程能力。特别是在调度算法的设计和实现过程中，我学会了如何将理论知识应用到实际问题中。此外，这次实验也让我意识到，在设计操作系统或类似系统时，算法的选择对系统性能有着重要影响。在未来的学习和工作中，我将更加注重理论与实践的结合，不断提高自己解决复杂问题的能力。