

进程管理项目电梯调度--项目说明文档

本项目是同济大学软件学院2018级操作系统课程设计项目

作者：徐炳昌

学号：1850953

指导教师：张慧娟

学院、专业：软件学院 软件工程

进程管理项目电梯调度--项目说明文档

1.项目需求分析

1.1 项目目的

1.2 项目需求

2.项目技术栈

2.1 开发环境

2.2 运行平台

2.3 python库

2.4 源码文件说明

3.项目主要逻辑功能与实现

3.1 数据结构

3.2 电梯状态及状态变化表

3.3 调度算法

4.项目界面及使用说明

4.1 项目界面

4.2 使用说明

1.项目需求分析

1.1 项目目的

- 通过控制电梯调度，实现操作系统调度过程；
- 学习特定环境下多线程编程方法
- 学习调度算法


1.2 项目需求

- 基本任务

1. 某个建筑一共有20层楼
2. 一共有五部互联的电梯

- 具体要求

1. 电梯内部有以下按键：楼层对应的数字键、开门键、关门键、报警键。
2. 电梯内部能够显示当前电梯所在的楼层和状态。
3. 在每一层楼、五个电梯门前应该有对应的上行键和下行键，在底楼没有下行键，在顶楼没有上行键。✂ 本项目实现为：在底楼下行键无效，在顶楼上行键无效。

4. 五部电梯相互联结，即当一个电梯按钮按下去时，其它电梯相应按钮同时点亮，表示也按下去了。 本项目实现为：用一个按钮替代五个按钮，按下一个相当于按下五个，为了节省空间。
5. 电梯调度算法：
 - 所有电梯初始状态都在第一层
 - 每个电梯没有相应请求情况下，则应该在原地保持不动；
 - 电梯调度算法自行设计。




2.项目技术栈

2.1 开发环境







- 开发环境：Windows10 企业版
- 开发平台：PyCharm Professional 2019.3.4
- 开发语言：Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

2.2 运行平台

 理论上tkinter是跨平台库

-  Windows 10
-  macOS (未测试)
-  Linux (未测试)

2.3 python库

- tkinter ( python自带)
 -  用于创建图形界面
- pipenv ( 需要安装)
 -  用于创建虚拟环境来打包，尽可能减少exe的大小
- pyinstaller ( 需要安装)
 -  用于打包为exe文件

2.4 源码文件说明

- elevatorTest2.py
 -  主要文件
- tkutils.py
 -  包含一部分ui

3.项目主要逻辑功能与实现

 只简单介绍项目运行逻辑，详见代码，有详细注释。

3.1 数据结构

- 枚举类：State表示电梯状态，Direction表示电梯移动方向，两者组合表示电梯状态，用于控制电梯移动，详见电梯状态转化图

```
class State(Enum):
    STOP = 0
    RUNNING = 1
class Direction(Enum):
    STOP = 0
    UP = 1
    DOWN = 2
```

- 指令列表：是一个元素为字典的列表
inside元素为{"ele_part": ele_part, "ele_num": ele_num, "target_floor": target_floor}
outside元素为{"target_direction": target_direction, "current_floor": current_floor}
- 内外指令：内指令实时刷新处理，外指令如果得不到调度暂时保留在指令列表

```
inside_ask = []
outside_ask = []
```

- 电梯集类：用于存储五部电梯的状态和调度到的指令，并非每一部电梯是一个实体，而是五部电梯为同一个实体。因为ui界面已定，而且根据需求，也没有泛化的需要。
- 要特殊说明的是ele_set[ele_num]用于找到某部电梯对应的字典，该字典里包含的指令队列也是字典。

```
class ElevatorSet:
    # 对应五台电梯，这里不做泛化可以随意控制电梯台数是因为即使这里允许，UI界面也不允许
    ele_set = {
        1: {"inside_queue": {}, "outside_up_queue": {}, "outside_down_queue": {}},
        2: {"inside_queue": {}, "outside_up_queue": {}, "outside_down_queue": {}},
        3: {"inside_queue": {}, "outside_up_queue": {}, "outside_down_queue": {}},
        4: {"inside_queue": {}, "outside_up_queue": {}, "outside_down_queue": {}},
        5: {"inside_queue": {}, "outside_up_queue": {}, "outside_down_queue": {}},
    }

    def __init__(self):
        for ele_num in range(1, 6):
            self.ele_set[ele_num]["floor"] = 1 # 当前楼层
            self.ele_set[ele_num]["state"] = State.STOP # 电梯状态
            self.ele_set[ele_num]["direction"] = Direction.STOP # 电梯方向
            self.ele_set[ele_num]["asked"] = False # 是否有请求
            self.ele_set[ele_num]["open"] = False # 电梯门状态
            self.ele_set[ele_num]["stay"] = False # 是否被请求停留在原地
            for j in range(1, 21): # 给内、外上、外下指令队列赋初值
                self.ele_set[ele_num]["inside_queue"][j] = 0
                self.ele_set[ele_num]["outside_up_queue"][j] = 0
                self.ele_set[ele_num]["outside_down_queue"][j] = 0
```

- 线程锁：用于防止不同线程访问同一个数据结构的时候发生冲突。

```
mutex = threading.Lock() # 创建一把线程锁
```

- 本项目中还有类似如下结构的三个监听器：用于刷新UI界面、指令队列等任务，防止堵塞主线程，这三个线程之间都采用了线程锁，防止访问相同数据结构的时候发生数据冲突

```
def refresher():
    # 刷新指令队列以及电梯状态---调度指令
    def _refresh_ask():
        mutex.acquire()
        _refresh_inside_ask()
        _refresh_outside_ask()
        mutex.release()

    # 每0.1秒执行一次
    def _main():
        while True:
            time.sleep(0.1)
            _refresh_ask()

    t = threading.Thread(target=_main, name="refresher")
    t.daemon = True
    t.start()
```

3.2 电梯状态及状态变化表

本程序使用两个枚举类对应的状态集合表示电梯的运行状态，电梯运行的主逻辑函数 `_ele_move1()` 就是根据电梯状态之间的状态转换来构建的，想要看懂，需要先了解下表及下图。

S: STOP R: RUNNING

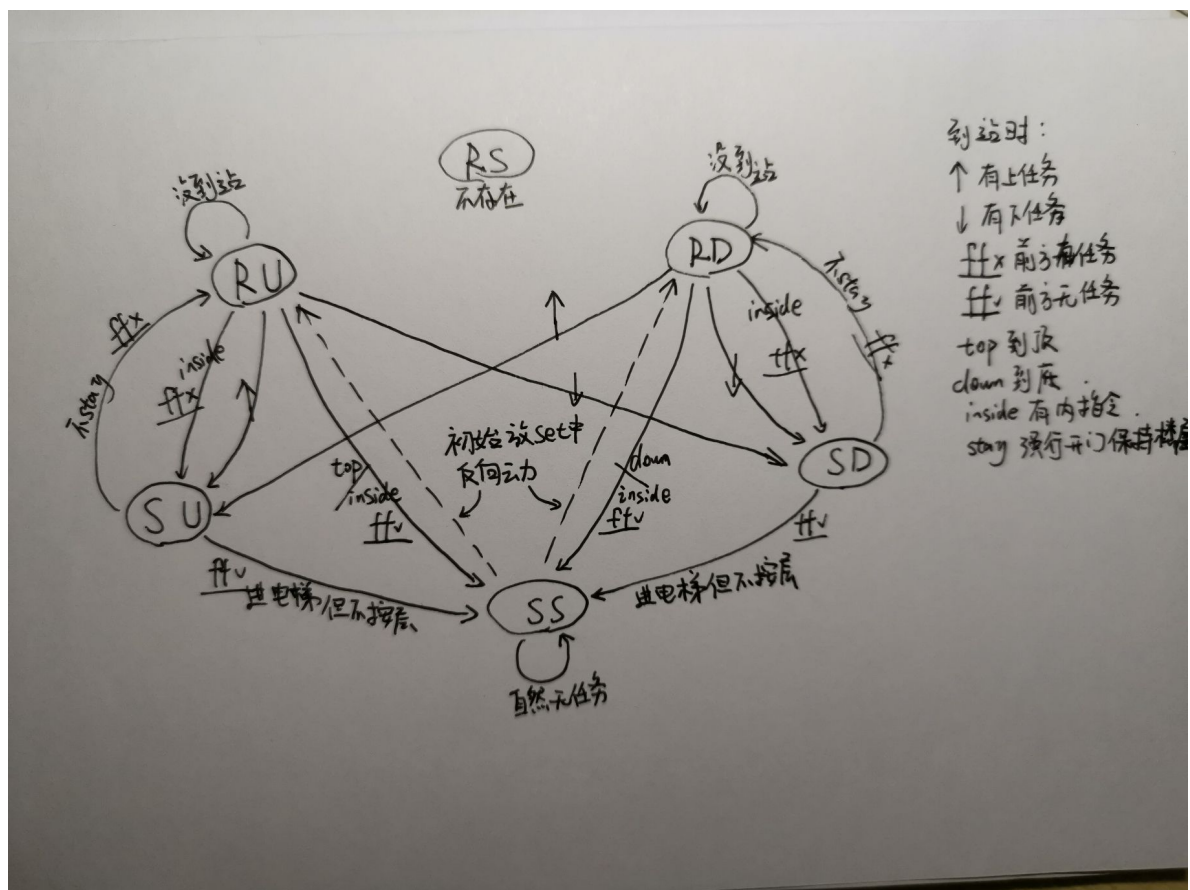
S: STOP U: UP D: DOWN

| 状态名 | 对应情况 |
|-----|----------------|
| SS | 电梯完全停止 |
| SU | 电梯当前停止，将继续向上移动 |
| SD | 电梯当前停止，将继续向下移动 |
| RU | 电梯当前运行，将继续向上移动 |
| RD | 电梯当前运行，将继续向下移动 |
| RS | 不存在 |

状态转换图：

🙄 找不到合适的软件画的出我想要的效果，只能手绘了

- 要注意的是，本项目中有监听器每两秒负责电梯移动，每次移动完成之后，都要根据下图转化关系，然后再进行下一次移动。有一点很特殊，就是初始时，电梯第一次收到指令的状态转化 `SS-->RU/RD`，是在生成指令的时候改变状态的，为了给电梯“初动力”。



3.3 调度算法

• 原理：

本算法类似LOOK算法，是一种按照楼层顺序依次服务请求的算法，它让电梯在最底层和最顶层之间连续往返运行，在运行过程中响应处在于电梯运行方向相同的各楼层上的请求，无论其是内请求还是外请求。所有的与电梯运行方向相同的乘客的请求在一次电梯向上运行或向下运行的过程中完成，免去了电梯频繁的来回移动以及有可能出现的饥饿问题。

算法发现电梯所移动的方向上不再有请求时立即停止或者响应另一个方向上的请求，避免移动到最底层或者最顶层时才改变运行方向，浪费时间。

• 实现：

- 当内部任意按钮或者外部任意楼层任意按钮被按下时，均生成对应的ask存放到inside_ask = [] 或者outside_ask = []中
- 每0.1秒执行一次的refresher()监听器会执行调度算法，将待处理的内外指令进行调度，送到各个电梯的存储内
- 内调度：按哪放哪，然后清楚待处理请求，不特殊调度
- 外调度：只有U和S状态的电梯会响应向上的请求，同理只有D和S状态的电梯会响应向下的请求。所有可响应的电梯将会存入possible_ele[ele_num]内，然后计算这些电梯和请求楼层间的距离，选择距离最近的电梯响应，响应后存入outside_XX_queue中，并清除该待处理指令。如果没有找到可以相应的电梯，该指令将被保留在待处理指令列表中，留待之后响应

| 变量 | 作用 |
|--------------------------|-------------------|
| inside_ask = [] 全局 | 存放待处理内指令的列表 |
| outside_ask = [] 全局 | 存放待处理外指令的列表 |
| "inside_queue": {} | 每部电梯被调度后存放的内部指令 |
| "outside_up_queue": {} | 每部电梯被调度后存放的外部向上指令 |
| "outside_down_queue": {} | 每部电梯被调度后存放的外部向下指令 |

4.项目界面及使用说明

4.1 项目界面

window 10 下的界面如下所示：

🔗tkinter库在不同平台下的样式不同，想要获得最佳的观感和体验请使用win10



- 下方五组按钮表示了对应的电梯内部的按钮，按下即表示该电梯内乘客想去哪一层。
- 右侧复选框切换选择对应的楼层，按上下等电梯按钮，相当于同时按下五部电梯的上下按钮。
- 下方五个方格对应五部电梯的显示屏，其显示该电梯当前楼层以及上下方向和开关门状态。

4.2 使用说明

请运行可执行目录下的dist目录下的exe文件

🤖如果运行失败，可以通过Pycharm打开源码部分的tkinter文件夹为项目，run即可，没有额外的库需求。