

电梯调度

操作系统第一次作业

1851521 沈天宇

项目介绍

本项目已托管在GitHub上，项目地址<https://github.com/Ultrasty/Elevator-Dispatching-Demo>

程序经过调试，除了偶尔会闪退之外没有bug！闪退可能是因为没用QThread而使用了python标准库自带的threading的原因，因为时间问题没有换过来。

1.背景

基本任务

某一层楼20层，有五部互联的电梯。基于线程思想，编写一个电梯调度程序。

功能描述

- ☑ 电梯应有一些按键，如：数字键、关门键、开门键、上行键、下行键、报警键等；
- ☑ 有数码显示器指示当前电梯状态；
- ☑ 每层楼、每部电梯门口，有上行、下行按钮、数码显示。

2.开发

- 使用python进行开发
- GUI开发使用的图形库为pyqt5
- python版本为3.8.1
- 过程式编程，电梯的状态使用一组全局变量表示，未给每部电梯设计一个类，线程间通过共享的这组全局变量进行通信，通过维护这组变量实现电梯的调度
- 运行：在根目录下运行如下命令（安装PyQt5和运行程序）

```
1 | pip install pyqt5
2 | python Elevator.py
```

- 或者运行打包好的.exe文件

```
1 | .\Elevator.exe
```

具体算法

过程式编程，电梯的状态使用一组全局变量elevator_goal、state、pause、floor表示，线程间通过共享的这组全局变量进行通信，通过维护这组变量实现电梯的调度，使用一个锁lock保证读写不冲突。

```
1 | # 表示目标楼层
2 |     elevator_goal1 = set([])
3 |     elevator_goal2 = set([])
4 |     elevator_goal3 = set([])
5 |     elevator_goal4 = set([])
6 |     elevator_goal5 = set([])
7 |     elevator_goal = [elevator_goal1, elevator_goal2, elevator_goal3,
8 |                       elevator_goal4, elevator_goal5]
9 |
10 |     # 此数组表示电梯状态 0表示停止 1表示向上运行 -1表示向下运行
11 |     state = []
12 |     for i in range(5):
13 |         state.append(0)
14 |
15 |     # 指示该电梯是否暂停运行
16 |     pause = []
17 |     for i in range(5):
18 |         pause.append(1)
```

```

18
19     # 表示当前楼层
20     floor = []
21     for i in range(5):
22         floor.append(1)
23
24     # 表示楼道里的向上的请求
25     people_up = set([])
26
27     # 表示楼道里的向下的请求
28     people_down = set([])
29
30     # 5个锁
31     lock = []
32     for i in range(5):
33         lock.append(threading.Lock())

```

当按下内部的按钮时，相应楼层会被添加至目标楼层

```

1 def set_goal(elev, flr): # 根据电梯内的按钮设定目标楼层
2     lock[elev - 1].acquire() # 获得锁
3     ex.findChild(QPushButton, "{0}+{1}".format(elev, flr)).setStyleSheet(
4         "QPushButton{background-image: url(background.png)}")
5     elevator_goal[elev - 1].add(flr)
6     lock[elev - 1].release() # 释放锁

```

当按下外部的按钮时，该楼层会被设为某一部电梯的任务楼层，该电梯即为距离该楼层最近的一部电梯。

```

1 def set_global_goal_up(flr): # 设定楼道里上楼请求所在的楼层
2     ex.findChild(QPushButton,
3         "up{0}".format(flr)).setStyleSheet("QPushButton{background-image:
4         url(background.png)}")
5     people_up.add(flr)
6     elevator_goal[
7         [abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] - flr),
8         abs(floor[3] - flr), abs(floor[4] - flr)].index(
9         min(abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] -
10        flr), abs(floor[3] - flr),
11        abs(floor[4] - flr)))]].add(flr)
12
13 def set_global_goal_down(flr): # 设定楼道里下楼请求所在的楼层
14     ex.findChild(QPushButton,
15         "down{0}".format(flr)).setStyleSheet("QPushButton{background-image:
16         url(background.png)}")
17     people_down.add(flr)

```

```

13     elevator_goal[
14         [abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] - flr),
15         abs(floor[3] - flr), abs(floor[4] - flr)].index(
16             min(abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] -
                flr), abs(floor[3] - flr),
                abs(floor[4] - flr)))]].add(flr)

```

上面两段代码是一样的，外部按钮的上楼和下楼区别在于，比如当外部按钮的▲10和▼10都被按下，而有一部电梯的升降路线为9→10→11则外部按钮的▲10会被清除而▼10会继续保持。此算法在函数check_and_change_floor中。

```

1     ...
2     # 从外部等候楼层中移除该层
3     if state[int - 1] == -1:
4         ex.findChild(QPushButton, "down{0}".format(floor[int -
5         1]))).setStyleSheet(
6             "QPushButton{}") # 移除标识
7         if state[int - 1] == 1:
8             ex.findChild(QPushButton, "up{0}".format(floor[int -
9             1]))).setStyleSheet(
10                 "QPushButton{}") # 移除标识
11     ...
12     ...
13     if state[int - 1] == 1:
14         people_up.discard(floor[int - 1]) # 移除楼层
15     if state[int - 1] == -1:
16         people_down.discard(floor[int - 1]) # 移除楼层
17     ...

```

五个线程每秒执行一次check_and_change_floor，进行更改当前楼层的操作，主要的算法也包含其中。

具体流程为先根据状态（上行、静止、下行）改变当前所在楼层，根据改变后的楼层从任务列表里去除相应楼层并且熄灭相应的按钮，再根据任务列表和当前所在的楼层改变电梯的状态。然后在睡眠1s之后，继续重复此流程。为每个电梯都安排一个这样的线程。

```

1 def check_and_change_floor(int):
2     while (1):
3
4         if pause[int - 1] == 1:
5             # 改变电梯楼层
6             lock[int - 1].acquire() # 加锁
7             if state[int - 1] == 0:

```

```

8         pass
9     else:
10         if state[int - 1] == -1:
11             floor[int - 1] = floor[int - 1] - 1
12         else:
13             floor[int - 1] = floor[int - 1] + 1
14         ex.findChild(QLCDNumber, "{0}".format(int)).display(floor[int -
15 1])
16         ex.findChild(QPushButton, "{0}+{1}".format(int, floor[int -
17 1])).setStyleSheet(
18             "QPushButton{}") # 去掉该层的标识
19
20         # 从外部等候楼层中移除该层
21         if state[int - 1] == -1:
22             ex.findChild(QPushButton, "down{0}".format(floor[int -
23 1])).setStyleSheet(
24                 "QPushButton{}") # 移除标识
25             if state[int - 1] == 1:
26                 ex.findChild(QPushButton, "up{0}".format(floor[int -
27 1])).setStyleSheet(
28                     "QPushButton{}") # 移除标识
29             if state[int - 1] == 1:
30                 if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int
31 - 1] in people_up):
32                     lock[int - 1].release()
33                     ex.findChild(QPushButton,
34 "open{0}".format(int)).setStyleSheet(
35                         "QPushButton{background-image: url(open.png)}")
36                     time.sleep(2)
37                     ex.findChild(QPushButton,
38 "open{0}".format(int)).setStyleSheet(
39                         "QPushButton{}")
40                     lock[int - 1].acquire()
41
42             if state[int - 1] == -1:
43                 if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int
44 - 1] in people_down):
45                     lock[int - 1].release()
46                     ex.findChild(QPushButton,
47 "open{0}".format(int)).setStyleSheet(
48                         "QPushButton{background-image: url(open.png)}")
49                     time.sleep(2)
50                     ex.findChild(QPushButton,
51 "open{0}".format(int)).setStyleSheet(
52                         "QPushButton{}")
53                     lock[int - 1].acquire()
54
55             if state[int - 1] == 1:
56                 people_up.discard((floor[int - 1])) # 移除楼层

```

```

47         if state[int - 1] == -1:
48             people_down.discard(floor[int - 1]) # 移除楼层
49         elevator_goal[int - 1].discard(floor[int - 1]) # 从要达到的目标楼
层中移除该层
50
51         # -----状态改变的算法----- #
52
53         if state[int - 1] == -1: # 如果当前状态是向下
54             if len(list(elevator_goal[int - 1])) == 0:
55                 state[int - 1] = 0
56             if (len(list(elevator_goal[int - 1])) != 0) and (
57                 min(list(elevator_goal[int - 1])) > floor[int -
1]):
58                 state[int - 1] = 1
59
60         if state[int - 1] == 1: # 如果当前状态是向上
61             if len(list(elevator_goal[int - 1])) == 0:
62                 state[int - 1] = 0
63             if (len(list(elevator_goal[int - 1])) != 0) and (
64                 max(list(elevator_goal[int - 1])) < floor[int -
1]):
65                 state[int - 1] = -1
66
67         if state[int - 1] == 0: # 如果当前状态是静止
68             if (len(list(elevator_goal[int - 1])) != 0) and (
69                 max(list(elevator_goal[int - 1])) > floor[int -
1]):
70                 state[int - 1] = 1
71             if (len(list(elevator_goal[int - 1])) != 0) and (
72                 min(list(elevator_goal[int - 1])) < floor[int -
1]):
73                 state[int - 1] = -1
74
75         # -----显示电梯楼层----- #
76         ex.findChild(QLCDNumber, "{0}".format(int)).display(floor[int -
1])
77
78         # -----间隔的时间----- #
79         lock[int - 1].release() # 释放锁
80         time.sleep(1)

```

其他功能

暂停

通过变量pause指示某部电梯是否暂停，若第i部电梯的pause值为0，则表示暂停运行

通过线程的if语句检查该变量，如果值为零，则什么都不做，然后time.sleep(1)

```
1 def check_and_change_floor(int):
2     while (1):
3         if pause[int - 1] == 1:
4             # 做事
```

OPEN

在电梯到达某一目标楼层时，显示OPEN，持续2s，该状态电梯所在楼层保持不变

通过sleep()来使线程进入睡眠，并且在进入睡眠前释放掉锁。

```
1         if state[int - 1] == 1:
2             if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int
- 1] in people_up):
3                 lock[int - 1].release()
4                 ex.findChild(QPushButton,
"open{0}".format(int)).setStyleSheet(
5                     "QPushButton{background-image: url(open.png)}")
6                 time.sleep(2)
7                 ex.findChild(QPushButton,
"open{0}".format(int)).setStyleSheet(
8                     "QPushButton{}")
9                 lock[int - 1].acquire()
10
11         if state[int - 1] == -1:
12             if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int
- 1] in people_down):
13                 lock[int - 1].release()
14                 ex.findChild(QPushButton,
"open{0}".format(int)).setStyleSheet(
15                     "QPushButton{background-image: url(open.png)}")
16                 time.sleep(2)
17                 ex.findChild(QPushButton,
"open{0}".format(int)).setStyleSheet(
18                     "QPushButton{}")
19                 lock[int - 1].acquire()
```

遇到的问题

程序在运行的时候有时候会崩溃，怀疑是因为多个线程同时读写一个变量导致的，于是让线程在读写要维护的全局变量的时候必须先获得一个锁，以进行线程间的同步。

```
1 # 设定5个锁
2 lock=[]
3 for i in range(5):
4     lock.append(threading.Lock())
```

需要读写系统要维护的变量时，必须先获得锁，读写完成后再释放锁

比如：（电梯每1s执行一次改变楼层的操作）

```
1 def check_and_change_floor(int):
2     while (1):
3         # 改变电梯楼层
4         lock[int-1].acquire() # 获得锁
5         ...
6
7         ...
8         lock[int-1].release() # 释放锁
9         time.sleep(1)
```

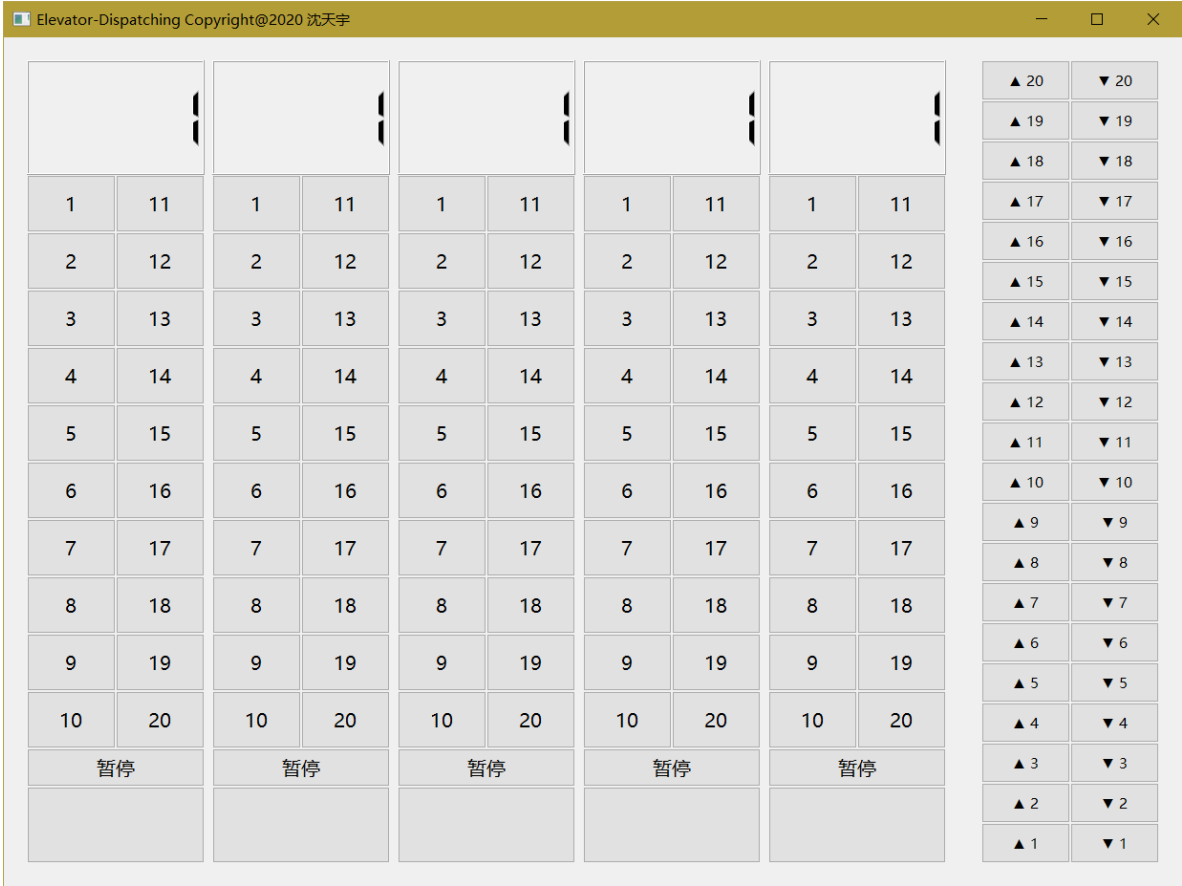
以及：（按钮按下时设定目标楼层）

```
1 def set_goal(elev, flr): # 设定目标楼层
2     lock[elev-1].acquire() # 获得锁
3     ...
4
5     ...
6     lock[elev-1].release() # 释放锁
```

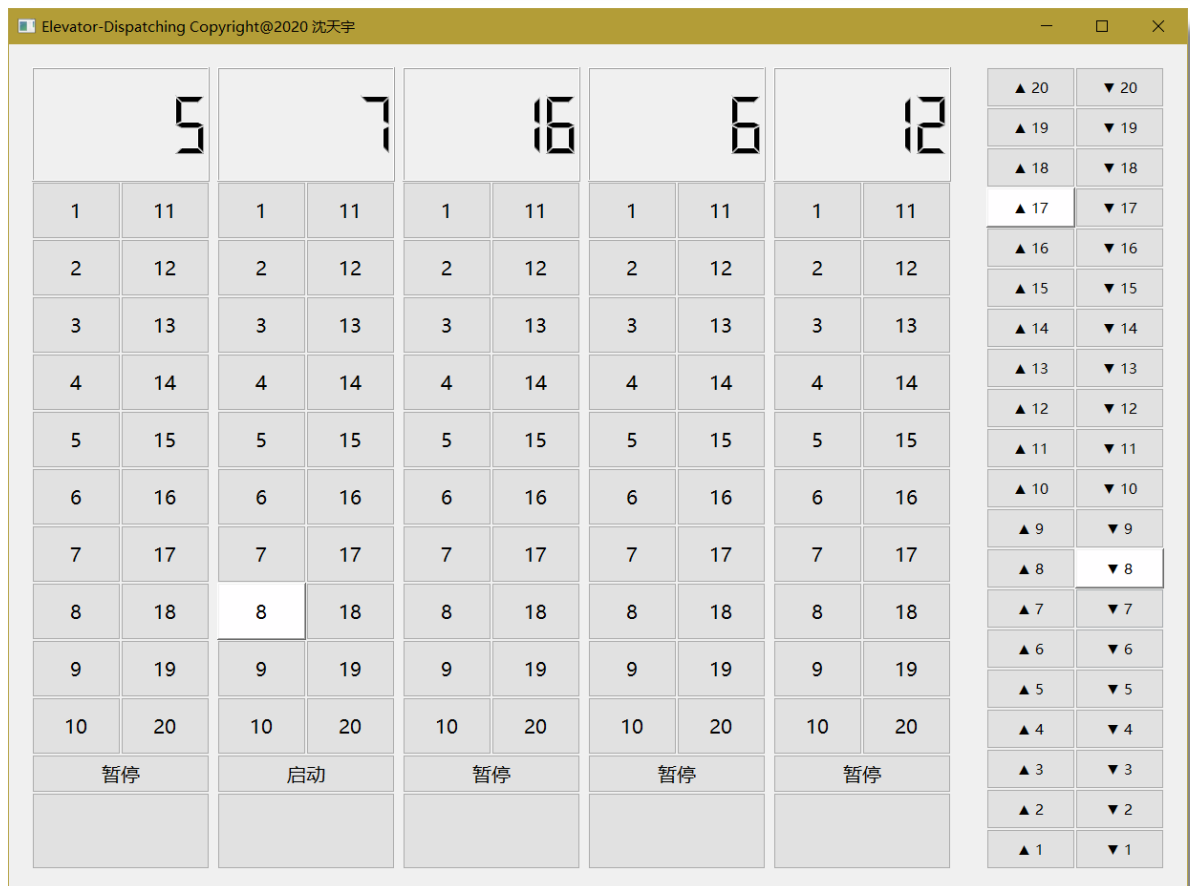
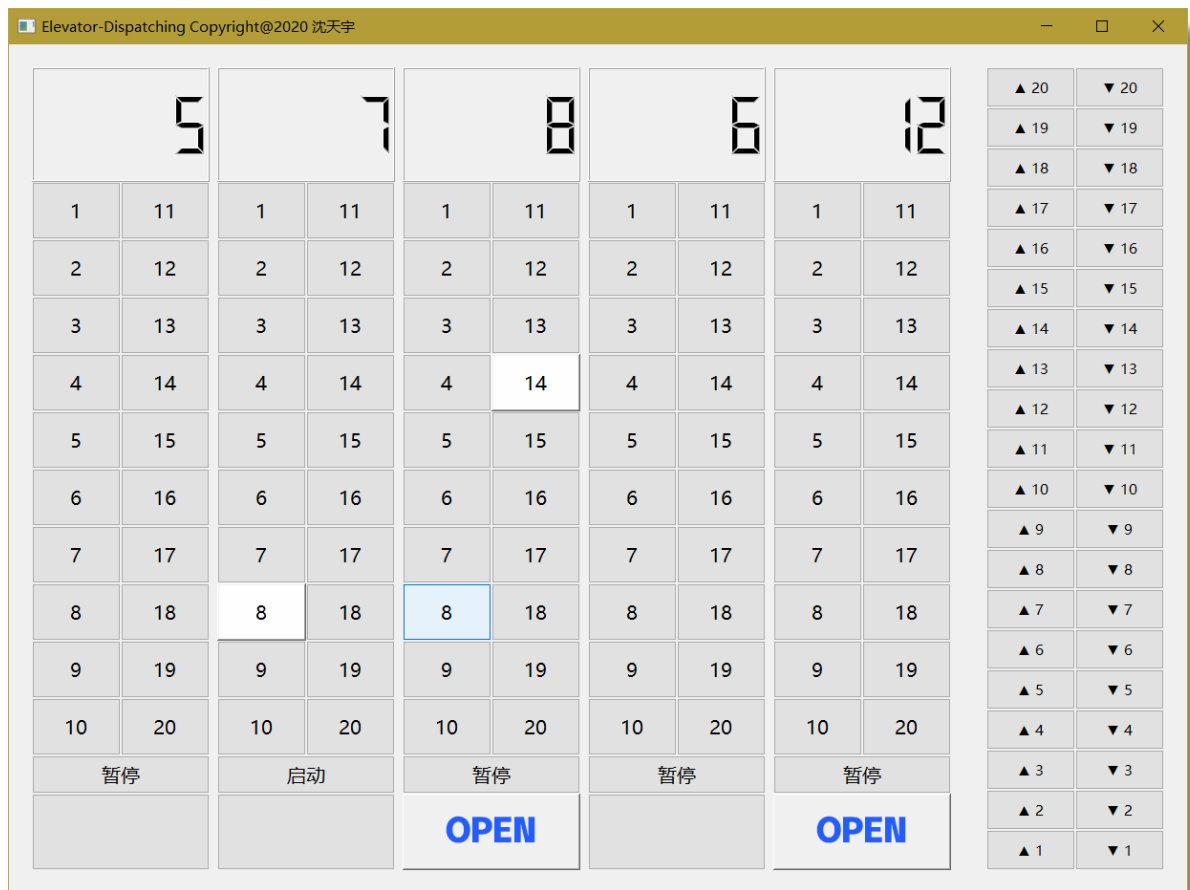
但是后来这个问题又出现了，百度谷歌退出时的错误代码也没能找到答案，仔细的一想了一下，可能是因为python标准库自带的threading和pyqt5不能很好的协作，因此每次运行都会出现一些随机性的问题，而pyqt5提供了一个QThread (PyQt5.QtCore.QThread) 可以提供多线程编程功能，在python自带多线程库的前提下，pyqt5还要有自己的多线程方法，可能就是为了更好的进行同步吧，但是由于时间原因，所以没有将threading换成QThread。

运行演示

- 初始状态



- 运行起来



更多的程序运行时详细信息可见我的录屏demo.mp4

心得体会

此次项目作业是使用python语言编写的，因为代码简单可以节省不少时间用来界面布局。因为之前基本都是用c++语言，所以花了一定的时间来熟悉python以及pyqt5。经过多次调试运行来修正代码，对于一个小型python项目有了一个简单的了解，并且对特定环境下多线程编程方法更加熟悉了。

参考书籍

因开发GUI需要用到PyQt5,因此参考了《PyQt5快速开发与实战》

另外还参考了PyQt5中文教程<https://github.com/maicss/PyQt5-Chinese-tutorial>!