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

- [x] 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；
- [x] 有数码显示器指示当前电梯状态；
- [x] 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）

```
pip install pyqt5
python Elevator.py
```

- 或者运行打包好的.exe文件

```
.\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。

```
# 表示目标楼层
    elevator_goal1 = set([])
    elevator_goal2 = set([])
    elevator_goal3 = set([])
    elevator_goal4 = set([])
    elevator_goal5 = set([])
    elevator_goal = [elevator_goal1, elevator_goal2, elevator_goal3, elevator_goal4, elevator_goal5]

    # 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
    state = []
    for i in range(5):
        state.append(0)

    # 指示该电梯是否暂停运行
    pause = []
    for i in range(5):
        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,
   "up{0}".format(flr)).setStyleSheet("QPushButton{background-image:
   url(background.png)}")
3      people_up.add(flr)
4      elevator_goal[
5          [abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] - flr),
   abs(floor[3] - flr), abs(floor[4] - flr)].index(
6              min(abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] -
   flr), abs(floor[3] - flr),
7                  abs(floor[4] - flr)))].add(flr)
8
9
10 def set_global_goal_down(flr):  # 设定楼道里下楼请求所在的楼层
11     ex.findChild(QPushButton,
   "down{0}".format(flr)).setStyleSheet("QPushButton{background-image:
   url(background.png)}")
12     people_down.add(flr)
```

```
    elevator_goal[
        [abs(floor[0] - flr), abs(floor[1] - flr), abs(floor[2] - flr), abs(floor[3] - flr), abs(floor[4] - flr)].index(
            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。

```
    ...
    # 从外部等候楼层中移除该层
    if state[int - 1] == -1:
        ex.findChild(QPushButton, "down{0}".format(floor[int - 1])).setStyleSheet(
            "QPushButton{}")  # 移除标识
    if state[int - 1] == 1:
        ex.findChild(QPushButton, "up{0}".format(floor[int - 1])).setStyleSheet(
            "QPushButton{}")  # 移除标识
    ...

    ...
    if state[int - 1] == 1:
        people_up.discard((floor[int - 1]))  # 移除楼层
    if state[int - 1] == -1:
        people_down.discard(floor[int - 1])  # 移除楼层
    ...
```

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。

```
def check_and_change_floor(int):
    while (1):

        if pause[int - 1] == 1:
            # 改变电梯楼层
            lock[int - 1].acquire()  # 加锁
            if state[int - 1] == 0:
```

```
                pass
            else:
                if state[int - 1] == -1:
                    floor[int - 1] = floor[int - 1] - 1
                else:
                    floor[int - 1] = floor[int - 1] + 1
            ex.findChild(QLCDNumber, "{0}".format(int)).display(floor[int - 1])
            ex.findChild(QPushButton, "{0}+{1}".format(int, floor[int - 1])).setStyleSheet(
                "QPushButton{}")  # 去掉该层的标识

            # 从外部等候楼层中移除该层
            if state[int - 1] == -1:
                ex.findChild(QPushButton, "down{0}".format(floor[int - 1])).setStyleSheet(
                    "QPushButton{}")  # 移除标识
            if state[int - 1] == 1:
                ex.findChild(QPushButton, "up{0}".format(floor[int - 1])).setStyleSheet(
                    "QPushButton{}")  # 移除标识
            if state[int - 1] == 1:
                if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int - 1] in people_up):
                    lock[int - 1].release()
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{background-image: url(open.png)}")
                    time.sleep(2)
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{}")
                    lock[int - 1].acquire()

            if state[int - 1] == -1:
                if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int - 1] in people_down):
                    lock[int - 1].release()
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{background-image: url(open.png)}")
                    time.sleep(2)
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{}")
                    lock[int - 1].acquire()

            if state[int - 1] == 1:
                people_up.discard((floor[int - 1]))  # 移除楼层
```

```
            if state[int - 1] == -1:
                people_down.discard(floor[int - 1])  # 移除楼层
            elevator_goal[int - 1].discard(floor[int - 1])  # 从要达到的目标楼层中移除该层

            # ----------------------状态改变的算法---------------------- #

            if state[int - 1] == -1:  # 如果当前状态是向下
                if len(list(elevator_goal[int - 1])) == 0:
                    state[int - 1] = 0
                if (len(list(elevator_goal[int - 1])) != 0) and (
                        min(list(elevator_goal[int - 1])) > floor[int - 1]):
                    state[int - 1] = 1

            if state[int - 1] == 1:  # 如果当前状态是向上
                if len(list(elevator_goal[int - 1])) == 0:
                    state[int - 1] = 0
                if (len(list(elevator_goal[int - 1])) != 0) and (
                        max(list(elevator_goal[int - 1])) < floor[int - 1]):
                    state[int - 1] = -1

            if state[int - 1] == 0:  # 如果当前状态是静止
                if (len(list(elevator_goal[int - 1])) != 0) and (
                        max(list(elevator_goal[int - 1])) > floor[int - 1]):
                    state[int - 1] = 1
                if (len(list(elevator_goal[int - 1])) != 0) and (
                        min(list(elevator_goal[int - 1])) < floor[int - 1]):
                    state[int - 1] = -1

            # ----------------------显示电梯楼层---------------------- #
            ex.findChild(QLCDNumber, "{0}".format(int)).display(floor[int - 1])
            # ------------------------间隔的时间------------------------ #
            lock[int - 1].release()  # 释放锁
        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)

```
def check_and_change_floor(int):
    while (1):
        if pause[int - 1] == 1:
        # 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。

```
            if state[int - 1] == 1:
                if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int - 1] in people_up):
                    lock[int - 1].release()
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{background-image: url(open.png)}")
                    time.sleep(2)
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{}")
                    lock[int - 1].acquire()

            if state[int - 1] == -1:
                if (floor[int - 1] in elevator_goal[int - 1]) or (floor[int - 1] in people_down):
                    lock[int - 1].release()
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{background-image: url(open.png)}")
                    time.sleep(2)
                    ex.findChild(QPushButton, "open{0}".format(int)).setStyleSheet(
                        "QPushButton{}")
                    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。

```
# 设定5个锁
    lock=[]
    for i in range(5):
        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）

```
def check_and_change_floor(int):
    while (1):
        # 改变电梯楼层
        lock[int-1].acquire() # 获得锁
        ...

        ...
        lock[int-1].release() # 释放锁
        time.sleep(1)
```

以及：（按钮按下时设定目标楼层）

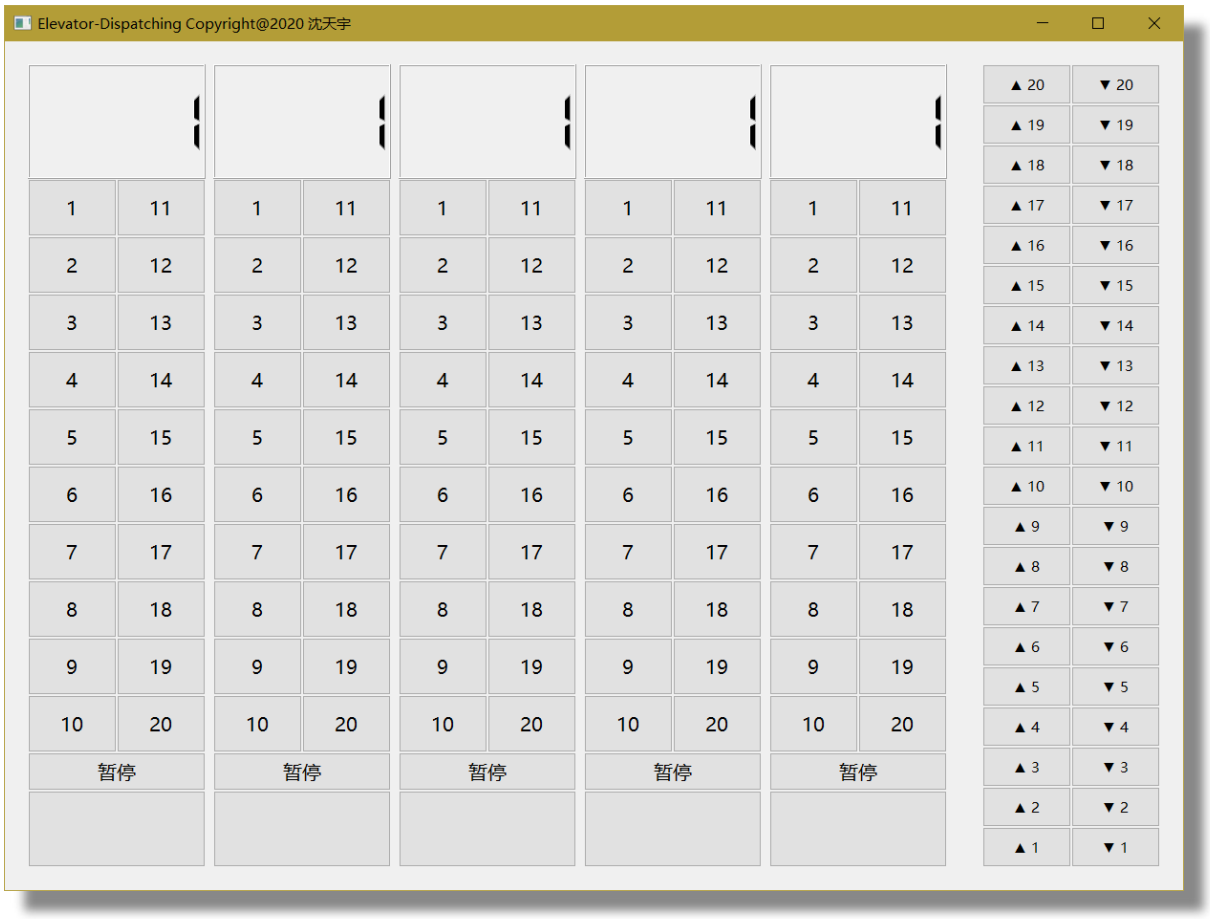
```
def set_goal(elev, flr):  # 设定目标楼层
    lock[elev-1].acquire() # 获得锁
    ...

    ...
    lock[elev-1].release() # 释放锁
```

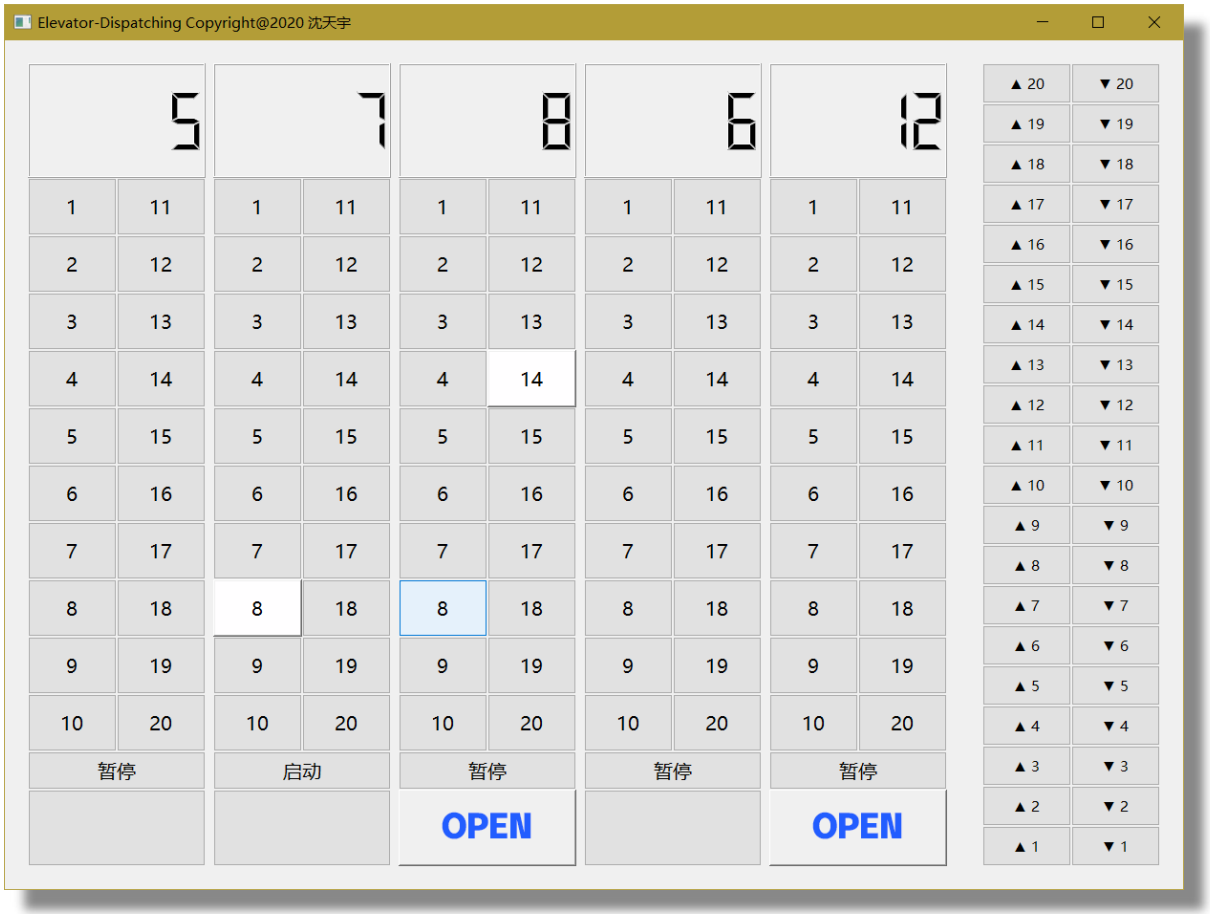
但是后来这个问题又出现了，百度谷歌退出时的错误代码也没能找到答案，仔细的想了一下，可能是因为python标准库自带的threading和pyqt5不能很好的协作，因此每次运行都会出现一些随机性的问题，而pyqt5提供了一个QThread（PyQt5.QtCore.QThread）可以提供多线程编程功能，在python自带多线程库的前提下，pyqt5还要有自己的多线程方法，可能就是为了更好的进行同步吧，但是由于时间原因，所以没有将threading换成QThread。

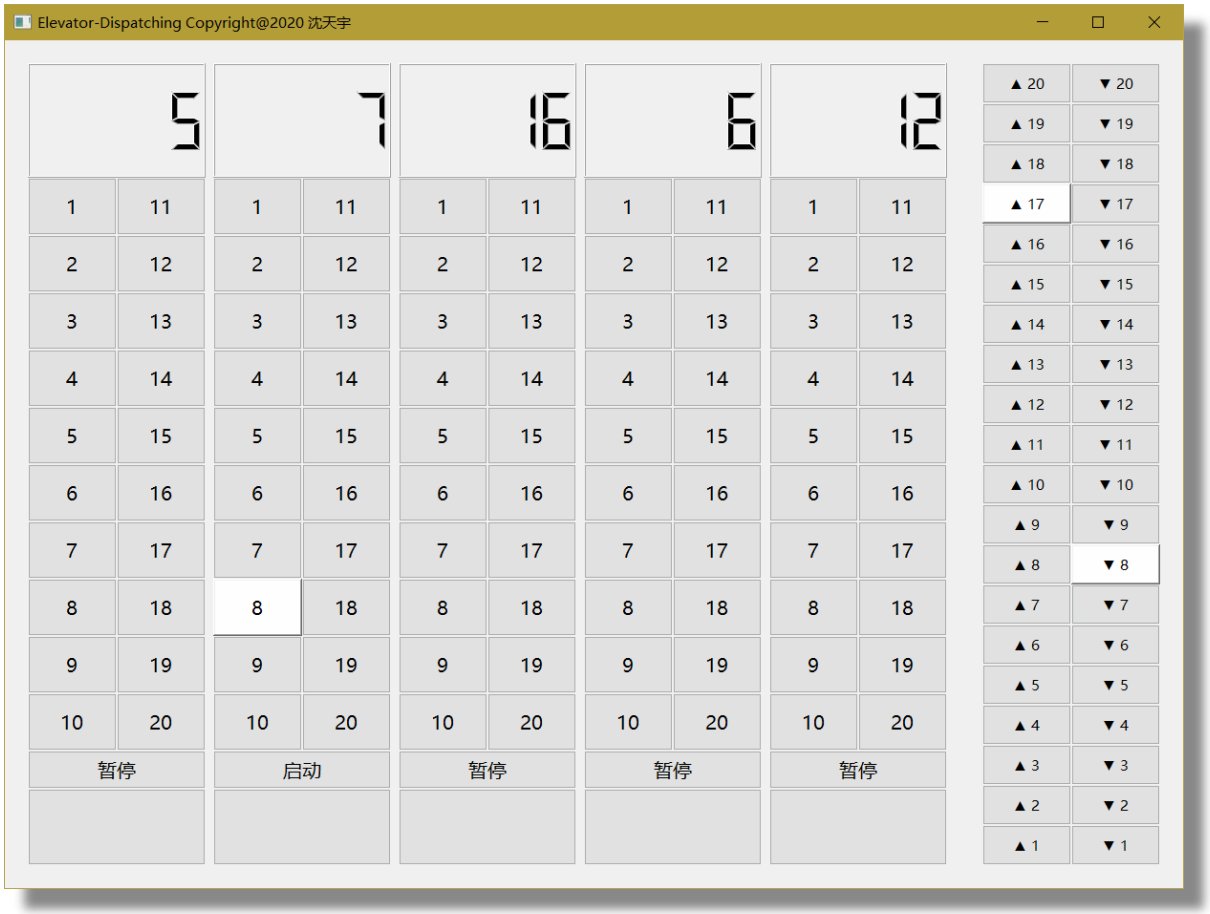
运行演示

- 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