学号：2151299

姓名：苏家铭

学院：软件学院

专业：软件工程

班号：42036902

指导老师：张惠娟

完成日期：2023年5月5日

电梯调度程序

[Operation System Project 1st]

[操作系统项目1]

[2023年5月5日]

目录

[1.项目介绍 2](#_Toc27260)

[1.1项目目的 3](#_Toc15459)

[1.2项目需求 3](#_Toc26679)

[2.开发环境 3](#_Toc4275)

[3.程序运行 3](#_Toc9773)

[3.1编译运行 3](#_Toc24529)

[3.2直接运行 4](#_Toc15965)

[4.功能实现 4](#_Toc21314)

[4.1前端界面 4](#_Toc18986)

[4.2选择电梯 7](#_Toc12059)

[4.3内部按钮被按下 7](#_Toc9213)

[4.4按下外部按钮 8](#_Toc6180)

[4.5报警（暂停使用） 9](#_Toc18813)

[4.6开关门 9](#_Toc14715)

[4.7电梯运行信息序列 11](#_Toc5394)

[5.算法 11](#_Toc6282)

[5.1全局变量 11](#_Toc16542)

[5.2数据结构与类 11](#_Toc4442)

[5.3功能逻辑 12](#_Toc25327)

[5.3.1对于电梯内部的按钮 12](#_Toc1733)

[5.3.2对于电梯外部的按钮 12](#_Toc26029)

[5.3.3电梯调度 12](#_Toc27943)

[5.3.4电梯层数改变 13](#_Toc8150)

[5.3.5 电梯内部按钮颜色改变 13](#_Toc19303)

[5.3.6 电梯状态检查 13](#_Toc24839)

[5.3.7改变电梯状态 14](#_Toc5144)

[5.3.8 锁 16](#_Toc17008)

# 1.项目介绍

## 1.1项目目的

1.1.1通过控制电梯调度，实现操作系统调度过程；

1.1.2学习特定环境下多线程编程方法

1.1.3学习调度算法

## 1.2项目需求

* 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

1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；

2有数码显示器指示当前电梯状态；

3每层楼、每部电梯门口，有上行、下行按钮、数码显示。

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。

5电梯调度算法；

1. 所有电梯初始状态都在第一层；
2. 每个电梯没有相应请求情况下，则应该在原地保持不动；
3. 电梯调度算法自行设计。

# 2.开发环境

* 开发环境：Windows 11
* 开发软件：Visual Studio code
* 开发语言：python(3.11.1)
* 主要引用快内容：PyQt5

# 3.程序运行

## 3.1编译运行

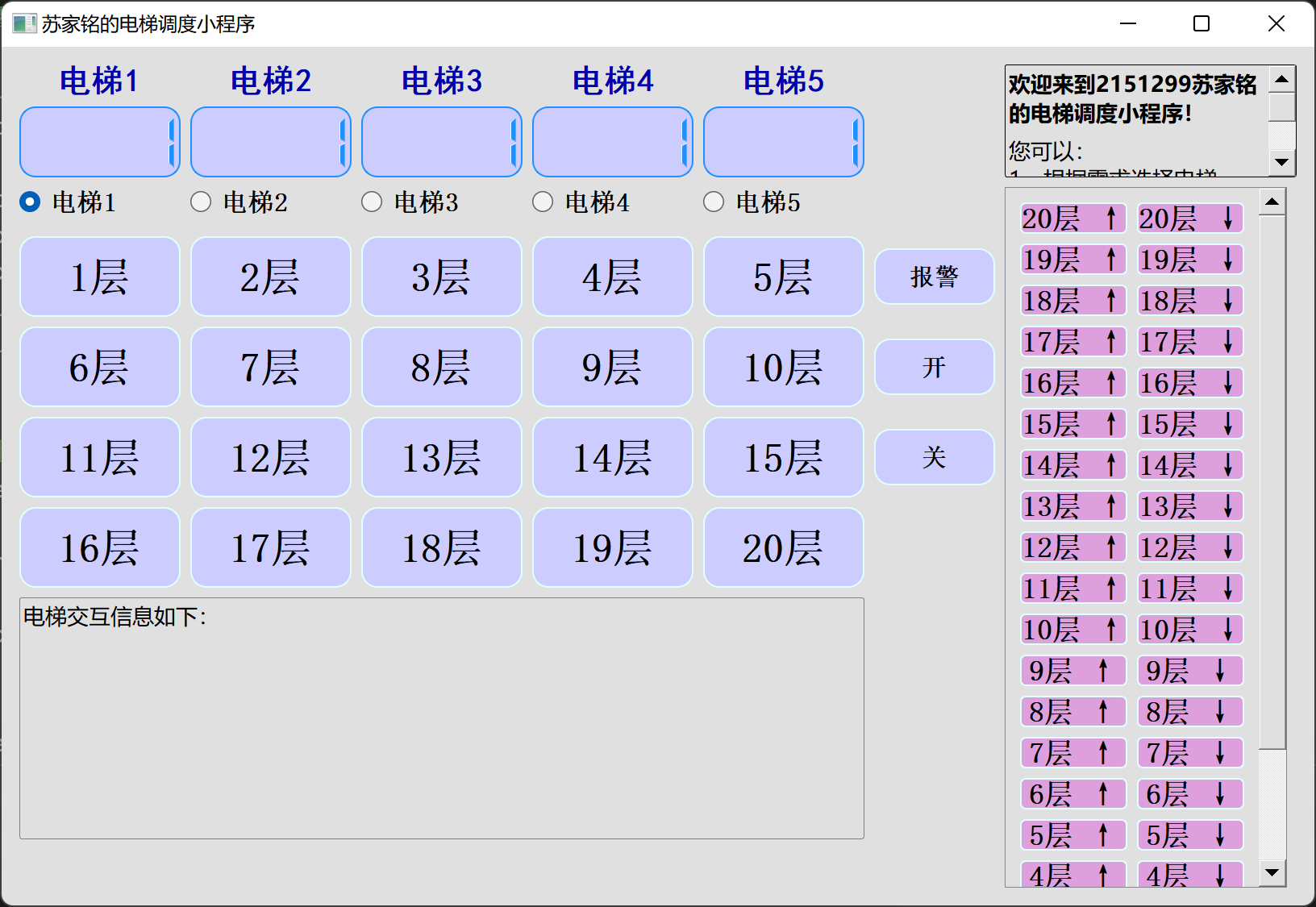
1. 安装PyQt5
2. 在命令行下输入pip install PyQt5
3. 切换至项目目录
4. 在命令行下输入python elevator01.py

3.2直接运行

在Windows系统下打开项目目录下的elevator01.exe可执行文件

# 4.功能实现

## 4.1前端界面



通过PtQt5代码实现：

grid = QGridLayout()

        # 电子显示管名称

        for i in range(5):

            word = QLabel('电梯' + str(i+1))

            word.setObjectName('name%d' % i)

            word.setStyleSheet("QLabel{border-radius:10px;border: 1px white; \

                                background-color:#e0e0e0;color:#0000AA; height:60; font-size:40px; \

                                font-family: SimHei;font-weight: 700;}")

            word.setAlignment(Qt.AlignCenter)

            grid.addWidget(word, 0, i)

        # 电子显示管 名称lcd0-4

        for i in range(5):

            lcd = QLCDNumber()

            lcd.setStyleSheet(lcd\_paint)

            # lcd.setFont(QFont("Consolas", 24, QFont.Bold))

            lcd.setObjectName('lcd%d' % i)

            # lcd.setSegmentStyle(QLCDNumber.Filled)

            grid.addWidget(lcd, 1, i, 2, 1)

        # 电梯选择按钮 名称rbtn0-4

        for i in range(5):

            rbtn = QRadioButton('电梯%d' % (i+1))

            rbtn.setStyleSheet('font-weight : bold;')

            rbtn.setFont(QFont('heiti', 12))

            rbtn.setObjectName('rbtn%d' % i)

            rbtn.clicked.connect(partial(radiobtn\_clicked, i))

            grid.addWidget(rbtn, 5, i)

            if i == 0:

                rbtn.setChecked(True)

        # 每个电梯的按钮 下方电梯名字btn0-19

        for i in range(20):

            btn = QPushButton('%d层' % (i+1))

            btn.setObjectName('btn%d' % i)

            btn.clicked.connect(partial(inner\_floor\_clicked, i))

            btn.setMinimumSize(200, 100)

            grid.addWidget(btn, (i // 5) + 7, i % 5)

        #报警键、开关键

        btn0 = QPushButton('报警')

        btn0.setObjectName('btn\_warning')

        btn0.clicked.connect(warning\_clicked)

        btn0.setMinimumSize(150, 70)

        btn0.setStyleSheet(warning\_paint)

        grid.addWidget(btn0, 7, 6, 1, 1)

        btn1 = QPushButton('开')

        btn1.setObjectName('btn\_open')

        btn1.clicked.connect(open\_clicked)

        btn1.setMinimumSize(150, 70)

        btn1.setStyleSheet(close\_open\_paint)

        grid.addWidget(btn1, 8, 6, 1, 1)

        btn2 = QPushButton('关')

        btn2.setObjectName('btn\_close')

        btn2.clicked.connect(close\_clicked)

        btn2.setMinimumSize(150, 70)

        btn2.setStyleSheet(close\_open\_paint)

        grid.addWidget(btn2, 9, 6, 1, 1)

        # 下方说明文字

        info = QTextEdit()

        info.setMinimumSize(100, 300)

        info.setObjectName('info')

        info.setFontPointSize(10)   # 定义格式字体大小

        info.setStyleSheet('border: 1px solid grey; border-radius:3px; font: 75 10pt "微软雅黑";')

        info.setText("电梯交互信息如下：")

        grid.addWidget(info, 11, 0, 9, 5)

        #右侧的说明文字

        # scrollArea = QScrollArea()

        # scrollArea.setWidgetResizable(True)  # 这条不加无法显示里面的控件

        note = QTextBrowser()

        note.setText("<h4>欢迎来到2151299苏家铭的电梯调度小程序！</h4> \

                    <h6>您可以：</h6><h6>1、根据需求选择电梯</h6><h6>2、左侧按下电梯内部楼层</h6> \

                    <h6>3、右侧按下电梯外部楼层</h6><h6>电子显示管显示了电梯此时的楼层信息</h6>")

        note.setStyleSheet('border: 1px solid black; border-radius:3px; font: 75 10pt "微软雅黑";')

        note.setObjectName('note')

        grid.addWidget(note, 0, 7, 3, 2)

        # 右侧的全局电梯按钮 名称floor\_up或down 0-19

        # grid -- scrollArea -w2- grid2

        grid2 = QGridLayout()

        scro = QScrollArea()

        w2 = QWidget()

        scro.setMinimumSize(350, 600)

        # scro.setStyleSheet('background-color: red')

        # w2.setStyleSheet('background-color: red')

        grid.addWidget(scro, 5, 7, 20, 1)

        for i in range(20):

            btn1 = QPushButton('%d层 ↑' % (20-i))

            btn2 = QPushButton('%d层 ↓' % (20-i))

            btn1.setObjectName('floor\_up%d' % (19-i))

            btn2.setObjectName('floor\_down%d' % (19-i))

            btn1.clicked.connect(partial(outer\_floor\_clicked, 'up', 19-i))

            btn2.clicked.connect(partial(outer\_floor\_clicked, 'down', 19-i))

            btn1.setStyleSheet(outer\_btn\_paint)

            btn2.setStyleSheet(outer\_btn\_paint)

            grid2.addWidget(btn1, i, 0)

            grid2.addWidget(btn2, i, 1)

        w2.setLayout(grid2)

        scro.setWidget(w2)

        self.setLayout(grid)

        # grid2 = QGridLayout()

        # grid.addLayout(grid2)

        self.setWindowTitle('苏家铭的电梯调度小程序')

        self.setStyleSheet('background-color: #e0e0e0')

鉴于本题的电梯按钮过多且重复性较高，所以本人舍弃了看似简易的“拖拽式”控件设计方式（使用Qt\_designer），而是采用直接编码，使用for语句来设计重复的按钮，达到化繁为简的效果。

## 4.2选择电梯



ELE\_NUM\_SELECTED = 0

通过一个全局变量ELE\_NUM\_SELECTED来指示此时用户选择的电梯号（按0-4编号）

当相应电梯前的按钮被按下，全局变量也随之改变。

# 事件：电梯选项按钮被点击 响应：修改全局ELE\_NUM\_SELECTED值

def radiobtn\_clicked(ele\_num):  # 参数为所选择的电梯号

    global ELE\_NUM\_SELECTED  # 声明全局变量才能修改

    ELE\_NUM\_SELECTED = ele\_num

    # print("ELEVATOR%d" % (ELE\_NUM\_SELECTED))

## 4.3内部按钮被按下



通过20个交互按钮可以选择电梯内部的楼层，悬浮、按下以及等待状态的电梯按钮都有相应的颜色表示。当内部电梯楼层被按下，inner\_goal\_floor[ele\_num]集合会记录该电梯的目标楼层。

# 事件：某部电梯内部楼层按键被点击 响应：1该电梯的目标楼层增加 2打印说明文字

def inner\_floor\_clicked(i):  # 参数为电梯号和所按楼层

    global ELE\_NUM\_SELECTED

    ele\_num = ELE\_NUM\_SELECTED

    inner\_goal\_floor[ele\_num].add(i)

    total\_goal\_floor[ele\_num].add(i)

    obj = sjm.findChild(QTextEdit, 'info')

    obj.append("电梯"+ str(ele\_num + 1) +"的内部楼层" + str(i+1) + "被按下！")

    # print("elevator%d goals are" % ele\_num)

## 4.4按下外部按钮



在右侧的滚轮框内可以选择按下相应的外部电梯楼层，与内部按钮一样，鼠标悬停、点击和按下后都会有相应的颜色表示。当按下外部楼层后，outer\_goal\_floor[up\_or\_down]集合会记录所按下的按钮信息，并且该楼层也会添加到电梯的总目标楼层集合total\_goal\_floor[ELE\_NUM\_SELECTED]中。

# 事件：外部电梯楼层按键被点击 响应：1目标楼层增加 2打印说明文字 3外部按钮变色

def outer\_floor\_clicked(up\_or\_down, i):  # 参数为上下和所按楼层

    global ELE\_NUM\_SELECTED

    # print('press floor\_{0}{1}'.format(up\_or\_down, i))

    # 添加外部目标数组

    if up\_or\_down == "up":

        outer\_goal\_floor[0].add(i)

    elif up\_or\_down == "down":

        outer\_goal\_floor[1].add(i)

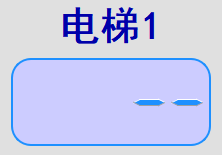
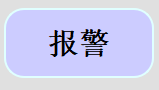
    total\_goal\_floor[ELE\_NUM\_SELECTED].add(i)

    obj = sjm.findChild(QTextEdit, 'info')

    obj.append("外部楼层" + str(i+1) + up\_or\_down + "被按下！")

    sjm.findChild(QPushButton, "floor\_{0}{1}".format(up\_or\_down, i)).setStyleSheet(outer\_btn\_paint\_2)  # 外部按钮变色

## 4.5报警（暂停使用）



当按下报警键后，该电梯的电子显示管会显示“--”以表示暂停使用，该电梯的所有目标楼层清零，外部楼层目标会被重新分配到没有故障的电梯。再次按下报警键即可让电梯重新运行。

# 事件：按下报警键 响应：1该电梯暂停状态被切换 2打印说明文字

def warning\_clicked():

    global ELE\_NUM\_SELECTED

    ele\_num = ELE\_NUM\_SELECTED

    obj = sjm.findChild(QTextEdit, 'info')

    if warning\_ele[ele\_num] == 0:  # 运行态

        warning\_ele[ele\_num] = 1  # 切换至暂停态

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "的报警键被按下，暂停使用！")

        print("elevator" + str(ele\_num + 1) + "stop")

    elif warning\_ele[ele\_num] == 1:  # 暂停态

        warning\_ele[ele\_num] = 0  # 切换至运行态

        print("elevator" + str(ele\_num + 1) + "run")

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "的报警键再次被按下，继续使用！")

    print(warning\_ele[ele\_num])

    # 自己总目标=0

    total\_goal\_floor[ele\_num] = set()

    # 外部目标再分配

    for i in range(5):

        if warning\_ele[i] == 0:

            for j in outer\_goal\_floor[0]:

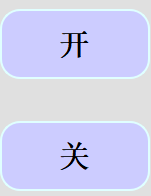
                total\_goal\_floor[i].add(j)

            for j in outer\_goal\_floor[1]:

                total\_goal\_floor[i].add(j)

            break

## 4.6开关门



本项目支持电梯开关门键，只有当电梯处于悬停状态才会在下方电梯运行信息框中显示开关门信息，而当电梯处于上行或下降过程中按下开关门键，下放信息框会显示错误信息。

# 事件：按下开门键 响应：1打印说明文字

def open\_clicked():

    global ELE\_NUM\_SELECTED

    obj = sjm.findChild(QTextEdit, 'info')

    if state\_ele[ELE\_NUM\_SELECTED] == 0:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "开门！")

    elif state\_ele[ELE\_NUM\_SELECTED] == 1:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "开门键被按下，但还在上升！")

    elif state\_ele[ELE\_NUM\_SELECTED] == -1:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "开门键被按下，但还在下降！")

# 事件：按下关门键 响应：1打印说明文字

def close\_clicked():

    global ELE\_NUM\_SELECTED

    obj = sjm.findChild(QTextEdit, 'info')

    if state\_ele[ELE\_NUM\_SELECTED] == 0:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "关门！")

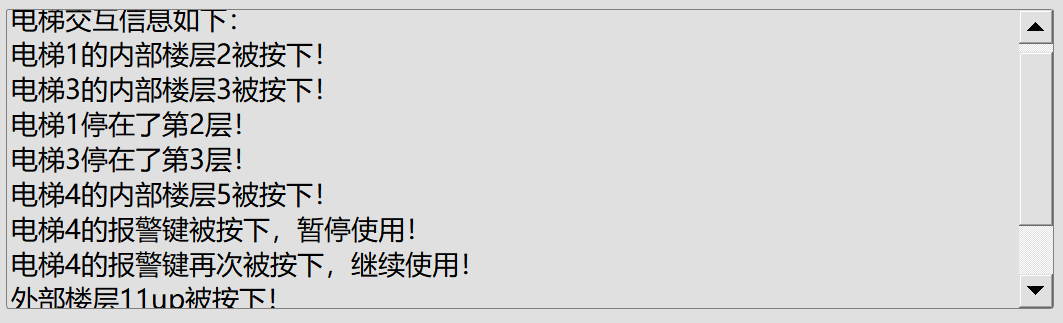
    elif state\_ele[ELE\_NUM\_SELECTED] == 1:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "关门键被按下，但还在上升！")

    elif state\_ele[ELE\_NUM\_SELECTED] == -1:

        obj.append("电梯" + str(ELE\_NUM\_SELECTED + 1) + "关门键被按下，但还在下降！")

## 4.7电梯运行信息序列



该信息框为一个QTextEdit对象，每当电梯发生不同的动作，都会有相应信息以类似如下的方式输出到该信息框，增强程序的人机交互性能。

obj = sjm.findChild(QTextEdit, 'info')

    obj.append("外部楼层" + str(i+1) + up\_or\_down + "被按下！")

# 5.算法

## 5.1全局变量

|  |  |  |
| --- | --- | --- |
| 编号 | 变量名 | 作用 |
| 1 | ELE\_NUM\_SELECTED | 记录此时用户选中的电梯号（编号从0开始） |
| 2 | now\_floor | 记录五个电梯此时的层 0-19 |
| 3 | inner\_goal\_floor | 记录五个电梯内部的目标层 0-19 |
| 4 | total\_goal\_floor | 记录五个电梯内部和外部的目标层 0-19 |
| 5 | outer\_goal\_floor | 记录外部电梯的目标层 0-19  outer\_goal\_floor[0]：上升  outer\_goal\_floor[1]：下降 |
| 6 | state\_ele | 记录电梯的状态 0表示悬停 1表示上升 -1表示下降 |
| 7 | stop\_ele | 记录五部电梯是否要停下 0:不停 1：要停 |
| 8 | warning\_ele | 记录每部电梯是否报警 0:正常运行 1:暂停 |

## 5.2数据结构与类

考虑到当电梯停在某一层时，若该层内部按钮和外部按钮都被按下，可以一起消去；并且为了解决用户点击多次相同按钮的问题，本项目使用了不可以有重复元素的数据结构——集合，而替换了传统的利用列表记录再排序的方法，节省了大量时间与空间消耗。

对于前端界面，本项目使用了sjm\_UI类来绘制，并且每个控件的交互通过若干全局函数和主函数中的变量进行实现。

## 5.3功能逻辑

### 5.3.1对于电梯内部的按钮

* 电梯楼层键被按下：把楼层作为该部电梯的目标加入到inner\_goal\_floor集合
* 报警键被按下：把该电梯的inner\_goal\_floor集合和total\_goal\_floor集合清空，把该电梯的外部outer\_goal\_floor目标集合重新分配到没有故障的电梯。
* 开关门键被按下：根据该电梯的报警情况，在信息框打印不同的反馈信息。

### 5.3.2对于电梯外部的按钮

* 外部电梯键被按下：
* 上行键：若该电梯故障则选择一个不故障的电梯，若该电梯不故障，把该楼层加入该电梯的outer\_goal\_floor[0]集合，同时也加入所选电梯的total\_goal\_floor集合。
* 下行键：若该电梯故障则选择一个不故障的电梯，若该电梯不故障，把该楼层加入该电梯的outer\_goal\_floor[1]集合，同时也加入所选电梯的total\_goal\_floor集合。

### 5.3.3电梯调度

一传统电梯调度算法：

* 先到先服务算法（First Come,First Served FCFS）。是一种随即服务算法，它不仅仅没有对寻找楼层进行优化，也没有实时性的特征，它是一种最简单的电梯调度算法。该算法根据乘坐电梯的先后次序调度，虽然公平、简单，但是一旦载荷较大，这种算法的性能会严重下降。
* 最短寻找楼层时间优先算法（Shortest Seek Time First SSTF）。该算法每次寻找可以最快满足的楼层（距离该电梯最近），这样可以减少平均响应时间较短，但响应时间的方差较大，原因是队列中的某些请求（例如20层的请求）可能长时间得不到响应，出现所谓的“饥饿”现象。
* 扫描算法（SCAN）：该算法是从移动臂当前位置开始沿着臂的移动方向去选择离当前移动臂最近的那个柱访问者，如果沿臂的移动方向无请求访问时，就改变臂的移动方向再选择。SCAN算法是电梯前进方向上的最短查找时间优先算法，它排除了电梯在盘面局部位置上的往复移动，SCAN算法在很大程度上消除了SSTF算法的不公平性，但仍有利于对中间磁道的请求。循环扫描算法（CSCAN）：规定电梯单向扫描，然后立即返回重新开始。
* LOOK 算法。该算法是对扫描算法的改进，当电梯发现移动方向上不再有目标时立即改变方向，无需移动到最底层或最高层。

本项目的电梯调度算法基于LOOK算法实现。但是当电梯发现前进方向上没有目标时会将状态改为悬停态而不是立即改变成反方向，直到该电梯有新的目标后才会依照目标产生新的方向。

### 5.3.4电梯层数改变

* 若电梯处于上升状态：层数+1
* 若电梯处于下降状态：层数-1

        if state\_ele[ele\_num] == 1:  # 向上状态

            now\_floor[ele\_num] += 1

        elif state\_ele[ele\_num] == -1:  # 向下状态

            now\_floor[ele\_num] -= 1

        else:  # 悬停状态

            pass

### 5.3.5 电梯内部按钮颜色改变

为了五个电梯共用一个该层内部按钮，所以对于绘制控件颜色的函数应该基于用户所选的电梯号。

        # 因为五个电梯共用一个该层内部按钮 所以应该按ELE\_NUM\_SELECTED分情况

        paint\_inner\_floor\_block(ELE\_NUM\_SELECTED)

### 5.3.6 电梯状态检查

当电梯处于a层：

1. 若电梯处于向上状态：

a在总目标层数中==>要停==>移除总目标中的a值+移除内部目标中的a值+移除外部向上目标中的a值

a不在总目标层数中==>继续向上

1. 若电梯处于向下状态：

a在总目标层数中==>要停==>移除总目标中的a值+移除内部目标中的a值+移除外部向下目标中的a值

a不在总目标层数中==>继续向下

1. 若电梯处于悬停状态：

a在总目标层数中==>要停==>移除总目标中的a值+移除内部目标中的a值+移除外部向上目标中的a值+移除外部向下目标中的a值

a不在总目标层数中==>继续悬停

        #  改变外部电梯颜色 + 停顿标识 + 移除目标

        if state\_ele[ele\_num] == 1:  # 向上

            if (now\_floor[ele\_num] in inner\_goal\_floor[ele\_num]) or (now\_floor[ele\_num] in outer\_goal\_floor[0]):

                stop\_ele[ele\_num] = 1  # 要停

            if (now\_floor[ele\_num] not in outer\_goal\_floor[1]):

                total\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除总目标

            inner\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除内部目标

            outer\_goal\_floor[0].discard(now\_floor[ele\_num])  # 移除外部目标

            sjm.findChild(QPushButton, "floor\_up%d" % (now\_floor[ele\_num])).setStyleSheet(outer\_btn\_paint)  # 外部按钮褪色

        elif state\_ele[ele\_num] == -1:  # 向下

            if (now\_floor[ele\_num] in inner\_goal\_floor[ele\_num]) or (now\_floor[ele\_num] in outer\_goal\_floor[1]):

                stop\_ele[ele\_num + 5] = 1  # 要停

            if (now\_floor[ele\_num] not in outer\_goal\_floor[0]):

                total\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除总目标

            inner\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除内部目标

            outer\_goal\_floor[1].discard(now\_floor[ele\_num])  # 移除外部目标

            sjm.findChild(QPushButton, "floor\_down%d" % (now\_floor[ele\_num])).setStyleSheet(outer\_btn\_paint)  # 外部按钮褪色

        elif state\_ele[ele\_num] == 0:  # 悬停

            if (now\_floor[ele\_num] in inner\_goal\_floor[ele\_num]) or (now\_floor[ele\_num] in outer\_goal\_floor[1]):

                stop\_ele[ele\_num + 5] = 1  # 要停

            elif (now\_floor[ele\_num] in outer\_goal\_floor[0]):

                stop\_ele[ele\_num] = 1  # 要停

            # if (now\_floor[ele\_num] not in outer\_goal\_floor[0]):

            total\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除总目标

            inner\_goal\_floor[ele\_num].discard(now\_floor[ele\_num])  # 移除内部目标

            outer\_goal\_floor[0].discard(now\_floor[ele\_num])  # 移除外部目标

            outer\_goal\_floor[1].discard(now\_floor[ele\_num])  # 移除外部目标

            sjm.findChild(QPushButton, "floor\_up%d" % (now\_floor[ele\_num])).setStyleSheet(outer\_btn\_paint)  # 外部按钮褪色

            sjm.findChild(QPushButton, "floor\_down%d" % (now\_floor[ele\_num])).setStyleSheet(outer\_btn\_paint)  # 外部按钮褪色

### 5.3.7改变电梯状态

1. 若电梯处于向上状态：

若总目标为空==>改为悬停状态

不满足上述情况时，若总目标不空且电梯在最大目标的上方==>改为下降状态

不满足上述情况时，若总目标不空且电梯到达最大的向下目标层==>改为悬停

1. 若电梯处于向下状态：

若总目标为空==>改为悬停状态

不满足上述情况时，若总目标不空且电梯在最小目标的下方==>改为上升状态

不满足上述情况时，若总目标不空且电梯到达最小的向上目标层==>改为悬停

1. 若电梯处于悬停状态：

若总目标为空==>改为悬停状态

不满足上述情况时，若总目标不空且电梯在最小目标的上方==>改为下降状态

不满足上述情况时，若总目标不空且电梯在最大目标的下方==>改为上升状态

        # 改变状态

        if state\_ele[ele\_num] == 1:  # 向上状态

            if len(list(total\_goal\_floor[ele\_num])) == 0 :  # 总目标已经空了 那么状态设为悬停

                state\_ele[ele\_num] = 0

            elif max(list(total\_goal\_floor[ele\_num])) < now\_floor[ele\_num]:  # 总目标不空 且 此时电梯在最大目标的上方

                state\_ele[ele\_num] = -1

            elif (max(list(total\_goal\_floor[ele\_num])) == now\_floor[ele\_num]):  # 总不空 且 最大外部向下等于此时电梯

                state\_ele[ele\_num] = 0

        elif state\_ele[ele\_num] == -1:  # 向下状态

            if len(list(total\_goal\_floor[ele\_num])) == 0:  # 总已经空了 就设为悬停

                state\_ele[ele\_num] = 0

            elif min(list(total\_goal\_floor[ele\_num])) > now\_floor[ele\_num]:  # 总不空 且 此时电梯在最小目标的下方

                state\_ele[ele\_num] = 1

            elif (min(list(total\_goal\_floor[ele\_num])) == now\_floor[ele\_num]):  # 总不空 且 此时最小外部向上等于此时电梯

                state\_ele[ele\_num] = 0

        else:  # 悬停状态

            if len(list(total\_goal\_floor[ele\_num])) == 0:  # 总已经空了 仍为悬停

                pass

            elif min(list(total\_goal\_floor[ele\_num])) < now\_floor[ele\_num]:  # 总不空 且 此时电梯在最小目标的上方

                state\_ele[ele\_num] = -1

            elif max(list(total\_goal\_floor[ele\_num])) > now\_floor[ele\_num]:  # 总不空 且 此时电梯在最大目标的下方

                state\_ele[ele\_num] = 1

### 5.3.8 锁

一组阻塞进程分别占有一定的资源并等待获取另外一些已经被同组其他进程所占有的资源时便会发生死锁，而每部电梯对于全局变量ELE\_NUM\_SELECTED和outer\_goal\_floor集合存在着竞争关系，若没有有效管理就有可能发生死锁现象。

在本项目中为了防止五个同时发生的线程对共有资源的读写冲突，可以采用两种方法避免：

* 利用QRadioButton选择按钮在用户层面上对ELE\_NUM\_SELECTED的进行选择，并且控制用户能且仅能选择一个选择按钮，每一时刻ELE\_NUM\_DELECTED的值是唯一的，而outer\_goal\_floor的使用必须基于ELE\_NUM\_SELECTED，也就是说限定了同一时刻只能有一步电梯使用或修改outer\_goal\_floor集合。
* 利用锁。

使用5个锁，在使用共有资源的前后使用锁来保证读写不冲突。

def check(ele\_num):

    while(True):

        lock[ele\_num].acquire()  # 要利用资源，获得锁

        ...

        # 对共有资源的读写操作

        ...

        lock[ele\_num].release()  # 使用完资源，释放锁