



# 同济大学

Tongji University

## 《操作系统》 项目报告I

报告名称: 处理机管理项目之电梯调度

班 级: 操作系统02班

学 号: 2253744

姓 名: 林觉凯

指导老师: 张惠娟

完成日期: 2024年4月24日

# 目录

1.项目介绍.....	3
1.1 项目目的.....	3
1.2 项目需求.....	3
1.3 开发环境介绍.....	3
2.项目运行.....	3
2.1 编译运行.....	4
2.2 直接运行.....	4
3.项目设计.....	4
3.1 项目多线程设计.....	4
3.2 项目前端设计.....	5
3.3 项目算法设计.....	7
4.项目功能.....	10
4.1 电梯外部按键.....	10
4.2 电梯内部按键.....	10
4.3 电梯开关门按键.....	10
4.4 电梯报警按键.....	11
4.5 电梯介绍文本.....	12
4.6 电梯信息文本.....	13
4.7 电梯数码管显示和标签.....	13
5.项目总结.....	14
5.1 项目遇到的问题.....	14
5.2 项目心得与收获.....	15

# 1.项目介绍

## 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

某一层楼 20 层，有五部互联电梯。基于线程思想，编写一个电梯调度程序。

- 功能描述

电梯应该有一些按键,比如：数码数字显示键、电梯关门键、电梯开门键、内部外部上行键、内部外部下行键、电梯报警键等；

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。

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。

- 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。

## 1.3 开发环境介绍

- 开发环境：Windows 11
- 开发软件：Visual Studio Code
- 开发语言：python 3.9.13 通过 conda 配置 python 环境，并且 pip 安装 PyQt5

# 2.项目运行

## 2.1 编译运行

若需要编译运行，需要安装 PyQt5。打开 Visual Studio Code，在终端命令行中输入 `pip install PyQt5`(如下图所示)。等待安装好 PyQt5 后，即可在 Visual Studio

Code 中编译运行 `os_project1_elevator.py` 并观察运行结果。



图 2.1 安装 PyQt5

## 2.2 直接运行

在 Windows 系统下直接打开文件夹中 `os_project1_elevator.exe` 即可运行。

## 3. 项目设计

### 3.1 项目多线程设计

本次项目使用 PyQt5 库中的 QThread 来实现多线程编程，QThread 是 PyQt 中用于创建多线程应用程序的类，它是基于 Qt 的 QThread 类的 Python 封装。使用 QThread，可以在应用程序中执行耗时的任务而不会阻塞用户界面，从而提高程序的响应性。使用 PyQt5 库中的 QThread 来实现多线程编程通常包括以下步骤：

- 创建自定义的线程类，本次项目定义了一个继承于 QThread 的电梯线程类 Elevator\_Thread 来进行电梯线程的代码编写。

- 实现线程所需要执行的任务，本项目在自定义线程类 Elevator\_Thread 中的“run()”中编写线程执行时所需要的相应任务代码。这些代码将在单独的线程中执行，并且不会阻塞主线程的运行。

- 使用信号和槽机制。在自定义线程类中定义需要发射的信号，并在适当的时候通过 emit() 方法发射信号。主线程可以连接到这些信号，并在接收到信号时执行相应的槽函数，进行线程间的通信。本项目在自定义电梯线程时实例化一个信号对象 update\_signal，并且通过 self.update\_signal.emit(self.elevator\_num) 将信号发射到槽函数 Elevator\_Update 中进行通信更新。

- 启动线程，在创建自定义线程类的实例后，本项目调用 start() 方法启动调度线程。电线程一旦启动，它将会执行其 run() 方法中的任务，完成项目的运转。

- 线程的生命周期管理，如果有需要，我们可以使用 wait() 方法等待线程完成，或者使用 isFinished() 方法检查线程是否已完成。可以在需要时中止线程的执行，也可以处理线程完成时的清理工作。

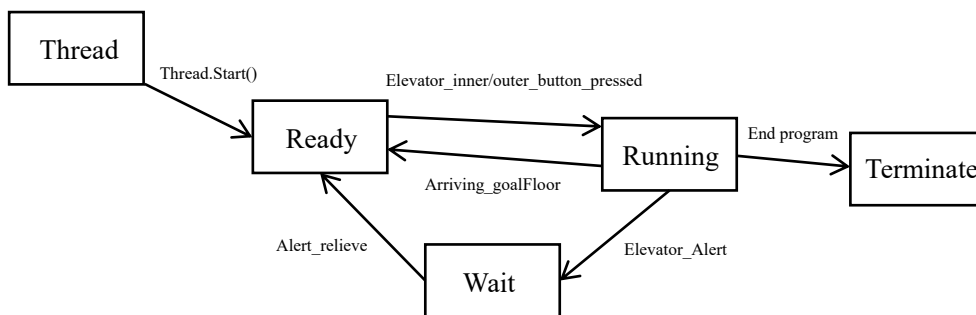


图 3.1.1 线程状态变化

## 3.2 项目前端设计

本项目使用的图形化界面设计是基于 PyQt5 完成的。主界面如下：

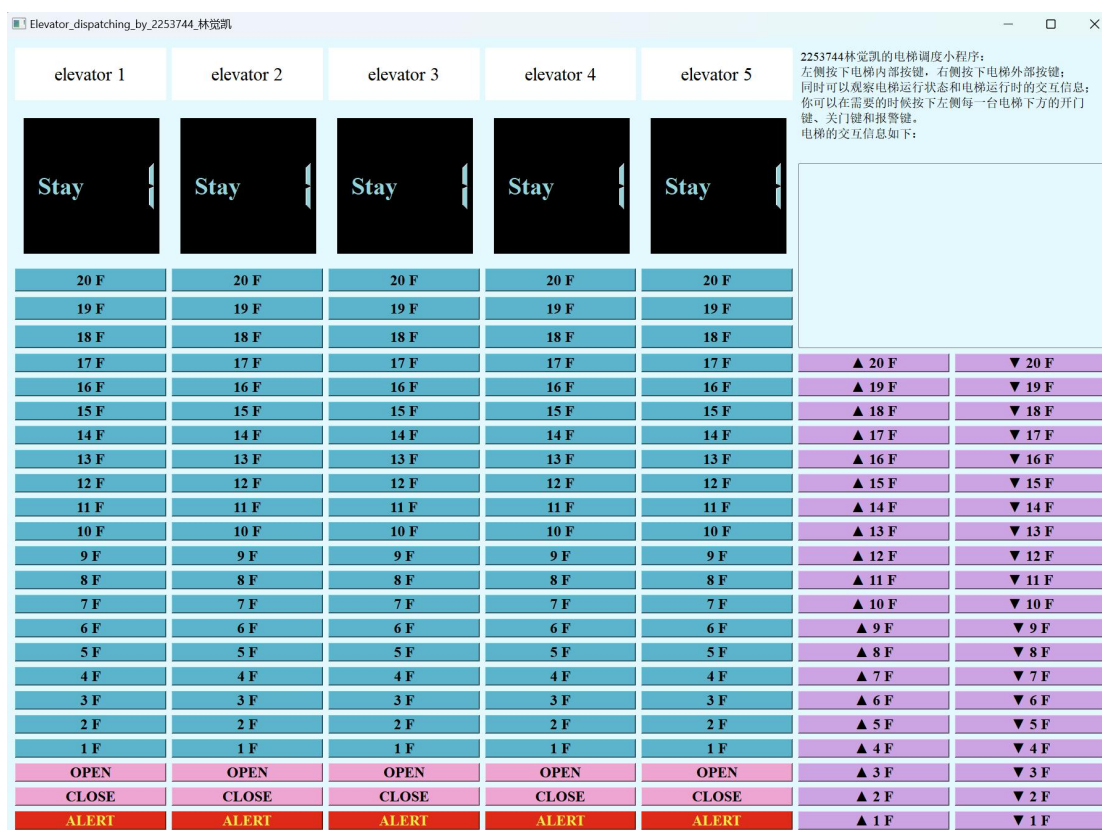


图 3.2.1 项目主界面

前端界面的控件主要分为以下几种：

**QPushButton:** 用于创建按钮，按钮通常用于触发特定的操作或事件，比如点击按钮执行某个函数或打开一个新窗口。本项目中的电梯内部、电梯外部按键和电梯开门关门报警按键都是这种控件类型，电梯内部按键是模拟用户进入某台电梯后电梯里面的楼层按键，电梯外部按键是某一层所在用户输出的请求。

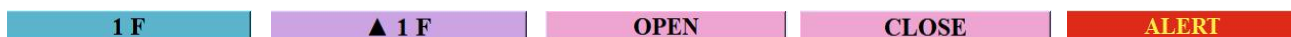


图 3.2.2 电梯内部按键、电梯外部按键、开门键、关门键、报警键

QLabel:用于显示文本或图像。它可以用来显示静态文本、HTML 格式的文本、图像等。本项目中的电梯标号和电梯状态标签属于这种控件类型。



图 3.2.3 电梯标号和电梯的状态标签

QLCDNumber:用于显示数字，类似于数字显示器或计时器。它通常用于显示整数值，例如计数器、计时器、测量值等。本项目中每部的电子数码管就是属于这种控件类型，通过每秒更新显示不同的数字来模仿电梯上下行的过程和电梯的所在楼层，直观地给用户以电梯运行的实时变化。



图 3.2.4 数字显示

2253744林觉凯的电梯调度小程序：  
左侧按下电梯内部按键，右侧按下电梯外部按键；  
同时可以观察电梯运行状态和电梯运行时的交互信息；  
你可以在需要的时候按下左侧每一台电梯下方的开门键、关门键和报警键。  
电梯的交互信息如下：

图 3.2.5 程序说明

QTextBrowser:用于显示富文本内容，类似于网页浏览器中的文本浏览器。与 QTextEdit 不同的是，QTextBrowser 主要用于显示静态文本，用户不能对其进行编辑，但可以包含超链接等交互元素。本项目中位于右上角的程序说明文本无法被编辑，故其就属于这种控件类型

QTextEdit:用于显示和编辑多行文本。它提供了丰富的功能，包括文本格式化、插入图片、设置字体样式、支持超链接等。本项目中实时输出电梯的交互信息，包括电梯的任务分配、电梯的开关门情况、电梯的报警情况和电梯内部的互相任务分配。项目中的交互信息输出文本框就属于这种控件类型。

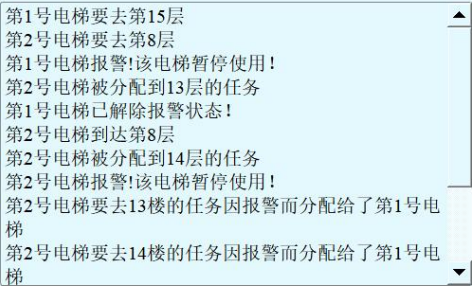


图 3.2.6 电梯交互信息

本项目的前端页面设计控件中具有很多有规律、重复的按钮设置，故而我直接采用 python 编代码的方式，利用循环和网格页面布局进行编辑。同时每一种按钮还有多种情况的颜色、标签的显示情况，这些是通过直接的代码编写来完成的(比如字体大小、字体类型、字体颜色、背景颜色、标签显示内容和控件之间的间隔等等)，这些具体的按钮不同状态的不同格式类型将在下文具体介绍，利用代码直接进行前端页面设计，使得修改、debug 变得更加方便。

### 3.3 项目算法设计

电梯调度算法是指根据乘客的请求和电梯的状态，合理地安排电梯的运行，以提高运行效率和服务质量。常见的电梯调度算法有如下几个：先来先服务算法(FCFS)：当一个请求到达时，就按照请求的顺序依次服务。简单直观，但是可能导致某些乘客等待时间过长。最短寻找时间优先算法(SSTF)：选择距离最近的电梯响应请求，以减少乘客的等待时间。可以提高系统的响应速度，但可能会出现饥饿现象，即距离较远的请求可能长时间得不到服务。扫描算法(SCAN)：电梯在一端到达，然后按照一个方向依次服务请求，直到到达另一端，然后改变方向继续服务请求。可以平衡电梯在两端的等待时间，但可能会导致一些请求长时间得不到服务。扫描算法(SCAN)：电梯在一端到达，然后按照一个方向依次服务请求，直到到达另一端，然后改变方向继续服务请求。可以平衡电梯在两端的等待时间，但可能会导致一些请求长时间得不到服务。LOOK 算法是一种电梯调度算法，它是扫描算法的一种变体。LOOK 算法的特点是电梯在扫描时不需要到达最低或最高楼层再返回，而是根据当前服务请求的方向，在相应方向上一直扫描直到没有请求为止，然后改变方向继续扫描，直到所有请求都被服务完毕。

本项目综合上述算法特点，设计了以下电梯调度算法：

首先定义一系列电梯数组：

#每部电梯是否报警：Elevator\_Alert = []

#每部电梯当前所在楼层：Elevator\_Floor = []

#每部电梯门是否该开：Elevator\_Should\_open = []

#每部电梯的目标楼层：Elevator\_Target = []

#每部电梯的当前状态：-1 下行，0 停留，1 上行：Elevator\_State = []

#电梯外部任务请求：Elevator\_Outer = set([])

没有报警的电梯的状态更新主要由以下代码执行：

```
if not Elevator_Alert[elevator_num - 1]:
    #如果电梯处于停留状态 0(没有上下行任务)
    if Elevator_State[elevator_num - 1] == 0:
        pass
    #如果电梯处于下行状态-1
    elif Elevator_State[elevator_num - 1] == -1:
    #如果这时已经到达最低层一层
        if Elevator_Floor[elevator_num - 1] == 1:
    #设置当前电梯楼层为 1
        Elevator_Floor[elevator_num - 1] = 1
    #电梯下行一层
else:
    Elevator_Floor[elevator_num - 1] -= 1
    #如果电梯处于上行状态 1
    else:
    #如果这时已经到达最高层 20 层
        if Elevator_Floor[elevator_num - 1] == 20:
    #设置当前电梯楼层为 20
            Elevator_Floor[elevator_num - 1] = 20
    #电梯上行一层
        else:
            Elevator_Floor[elevator_num - 1] += 1
```

这段代码主要用于正常电梯的楼层的维护，通过对电梯状态的判断，即向上运行、停留和向下运行的状态，更新下一秒电梯的所在楼层。

上段代码是对于可视化电梯楼层更新的重点，我们发现，电梯的楼层主要是由当前电梯的状态所决定的，电梯的状态在这就显得尤为重要。电梯状态的判断也是该算法的关键。以下是对电梯状态的分析算法：

```
#如果当前电梯楼层对于该电梯的目标楼层或
是电梯外部按钮的目标楼层
    if (Elevator_Floor[elevator_num - 1] in
Elevator_Target[elevator_num - 1] or
(Elevator_Floor[elevator_num - 1] in
Elevator_Outer)):
    #这时电梯应该开门
    Elevator_Should_open[elevator_num - 1] = 1
    #电梯外部楼层的楼层请求集中删去该层请求
    Elevator_Outer.discard(Elevator_Floor[elevator_num
- 1])
    #该电梯的请求楼层中删去该楼层请求
    Elevator_Target[elevator_num -
1].discard(Elevator_Floor[elevator_num - 1])
    #计算该电梯剩余楼层任务
    LeftFloors_to_go =
len(Elevator_Target[elevator_num - 1])
    HighestFloor_to_go = -9 if LeftFloors_to_go == 0
    else max(Elevator_Target[elevator_num - 1])
    LowestFloor_to_go = 999 if LeftFloors_to_go == 0
    else min(Elevator_Target[elevator_num - 1])
    #如果电梯正处于下行状态
    if Elevator_State[elevator_num - 1] == -1:
    #此时没有其他任务了
        if LeftFloors_to_go == 0:
            #将电梯状态设为停留状态
            Elevator_State[elevator_num - 1] = 0
        #将电梯状态设为停留状态
        Elevator_State[elevator_num - 1] = 0
    #如果剩余任务的楼层高于当前所在楼
    elif LowestFloor_to_go >
        Elevator_Floor[elevator_num - 1]:
    #电梯状态设为上行状态
        Elevator_State[elevator_num - 1] = 1
    #如果电梯正处于停留状态而且有目标任务
    if Elevator_State[elevator_num - 1] == 0
and LeftFloors_to_go != 0:
    #比较当前所在楼层和目标任务
    楼层大小决定上行还是下行
        if HighestFloor_to_go >
            Elevator_Floor[elevator_num - 1]:
            Elevator_State[elevator_num - 1] = 1
        elif LowestFloor_to_go <
            Elevator_Floor[elevator_num - 1]:
            Elevator_State[elevator_num - 1] = -1
    #如果电梯正处于上行状态
    if Elevator_State[elevator_num - 1] == 1:
    #此时没有其他任务了
        if LeftFloors_to_go == 0:
            #将电梯状态设为停留状态
            Elevator_State[elevator_num - 1] = 0
```



```

#如果剩余任务的楼层低于当前所在楼层
elif HighestFloor_to_go < Elevator_Floor[elevator_num - 1]:
    Elevator_Floor[elevator_num - 1]:
#电梯状态设为下行状态
Elevator_State[elevator_num - 1] = -1

```

内部电梯有一个目标数组，储存着该电梯的目标楼层。注意，在每个电梯到达相应的楼层之后，要将该目标楼层除去。每达到一个目标楼层后，这里我们使用 `LeftFloors_to_go` 变量来记录该电梯剩余目标楼层的数量，如果没有剩余任务就将状态设为 0(停留)，如果电梯正处于下行状态且剩余任务的楼层高于当前所在楼层，则电梯状态改为上行；如果电梯正处于上行状态且剩余任务的楼层低于当前所在楼层，则电梯状态改为下行；如果电梯正处于停留状态且有了任务，则电梯状态改为根据当前楼层于目标楼层的比较来决定改变为上行或下行状态。

当然，电梯楼层的状态也与电梯的报警状态有关，电梯的报警后算法如下：

```

#该电梯剩余外部楼层任务(可以认为是外部任务)
#分配给别的电梯,当前楼层内的内部任务根据常识
#无法分配
Elevator_Target[NewElevator].add(floor)
updateInfo =
MyWindow.findChild(QTextEdit, 'information')
if updateInfo is not None:
    updateInfo.append(f'第
{elevator_num}号电梯要去 {floor} 楼的任务因报警
而分配给了第 {NewElevator + 1} 号电梯")
print(f'第 {elevator_num} 号
电梯要去 {floor} 楼的任务因报警而分配给了第
{NewElevator + 1} 号电梯")
#将该报警电梯的外部楼层任务从该
电梯任务中删去
for LeftFloor in Leftneeded_Floor:
    Elevator_Target[elevator_num -
1].discard(LeftFloor)
if len(Elevator_Target[elevator_num -
1]) == 0:
    Elevator_State[elevator_num - 1] = 0
Leftneeded_Floor = set([])
for floor in Elevator_Outer:
    if floor in Elevator_Outer:
        #该数组存储剩余外部楼层任务和当前哪个
        #电梯距离相近
        Elevator_Floor_gap = []
        for elev in range(ELEVATOR_NUM):
            Elevator_Floor_gap.append(10000
            if Elevator_Alert[elev] else abs(Elevator_Floor[elev] -
            floor))
        #找到最近的电梯转移任务
        NewElevator =
Elevator_Floor_gap.index(min(Elevator_Floor_gap))
Leftneeded_Floor.add(floor)

```

以上是电梯原来是正常的然后变为警报状态的算法框架，(如果电梯是警报状态再按警报键为接触警报状态，改变按钮样式即可)。按照日常的思维，如果电梯此时外部按钮有分配给这台电梯，任务会被重新分配。该算法为寻找报警电梯外部安排的任务，对每一个正常电梯遍历，找楼层数差值最小值，然后将这个任务转移到新的电梯的任务中去，这样就完成了故障电梯任务的重新分配。

## 4.项目功能

### 4.1 电梯外部按键

电梯外部按键是在每一层外部的按键，我这里把电梯外部按键所发送的任务请求为电梯调度系统的外部公共任务。如下图所示，电梯外部按钮在没有操作的时候保持较深的紫色，而在鼠标悬浮放置和按下之后(一直到该层电梯外部楼层任务完成)保持较浅的紫色，这是由于我在设计外部电梯按钮时赋予了以下格式：

电梯外部按钮初始格式(深紫色)：

```
upbtn.setStyleSheet('QPushButton{background-color: #CCA4E3;color: black; font: bold 12pt "Times new roman";}QPushButton: hover{background-color:#E9D7DF}QPushButton:pressed{background-color:#E9D7DF}'
```

电梯外部按钮按下保持格式(浅紫色)：

```
ClickButton.setStyleSheet('QPushButton{background-color: #E9D7DF;color:black; font: bold 12pt "Times new roman";}')
```



图 4.1.1 电梯外部按钮及其按下格式

### 4.2 电梯内部按键

电梯内部按键是在每一台电梯内部的按键，即进入每台电梯后的按键。所分配到的任务为该电梯的内部任务。如下图所示，电梯内部按钮在没有操作的时候保持较深的蓝色，而在鼠标悬浮放置和按下之后(一直到该层电梯内部楼层任务完成)保持较浅的蓝色，这是由于我在设计内部电梯按钮时赋予了以下格式：

电梯内部按钮初始格式(深蓝色)：

```
inner_btn.setStyleSheet('QPushButton{background-color:#5CB3CC;color:black;font:bold12pt"Timesnewroman";}QPushButton: hover{background-color:#C3D7DF}QPushButton:pressed{background-color:#C3D7DF}')
```

电梯内部按钮按下保持格式(浅蓝色)：

```
ClickButton.setStyleSheet('QPushButton{background-color: #C3D7DF;color:black; font: bold 12pt "Times new roman";}')
```



图 4.2.1 电梯内部按钮及其按下格式

### 4.3 电梯开关门按键

电梯开关门按键在每一台电梯内部，按下电梯的开关门键之后会出现相应的

现象。如下图所示，电梯开关门按钮在没有操作的时候保持较深的粉色，而在鼠标悬浮放置和按下之后保持较浅的粉色，这是由于我在设计内部电梯按钮时赋予了以下格式：

电梯开关门按钮初始、悬浮和按下格式(粉色和浅粉色)：

```
open_btn.setStyleSheet('QPushButton{background-color: #EEA5D1;\color: black; font: bold 12pt "Times newroman";}"QPushButton: hover{background-color:#F3D3E7}"QPushButton:pressed{background-color:#F3D3E7}')
close_btn.setStyleSheet('QPushButton{background-color: #EEA5D1;\color: black; font: bold 12pt "Times newroman";}"QPushButton: hover{background-color:#F3D3E7}"QPushButton:pressed{background-color:#F3D3E7}')
```



图 4.3.1 电梯开关门按钮格式

电梯在运行的时候会进行电梯状态的判断，如果电梯正在运行，则不会开门，或者是已经关门；如果电梯正在报警，则不会开门和关门。

第1号电梯正在运行中，不开门	第2号电梯报警中，不开门
第1号电梯正在运行中，已关门	第2号电梯报警中，不关门

图 4.3.2 电梯开关门在特殊情况的输出文本显示

如果电梯在合适的情况下电梯可以开门，标签状态则会显示：

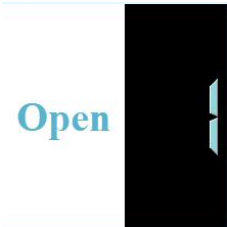


图 4.3.3 电梯在开门键按下后的标签显示

### 4.4 电梯报警按键

电梯报警按键在每一台电梯内部，按下电梯的报警键之后会改变电梯的状态，我们先来讨论一下电梯报警键的格式。如下图所示，电梯报警门按钮在没有操作的时候保持较深的红色，而在鼠标悬浮放置和按下之后(一直到该层电梯报警状态)保持较浅的红色，这是由于我在设计电梯报警按键时赋予了以下格式：

电梯报警按键按钮初始格式(深红色)：

```
alert_btn.setStyleSheet('QPushButton{background-color: #DE2A18;\color: #FFF143; font: bold 12pt "Times newroman";}"QPushButton: hover{background-color:#F0945D}"QPushButton:pressed{background-color:#F0945D}')
```

D}'))

电梯报警按键按钮按下保持格式(浅红色):

```
AlertButton.setStyleSheet("QPushButton{background-color:#F0945D;\color:#FFF143;font:bold 12pt "Times newroman";}"QPushButton:hover{background-color:#DE2A18}"QPushButton:pressed{background-color:#DE2A18}')"
```



图 4.4.1 电梯报警键及其按下格式

电梯报警之后电梯的状态标签会改变为"Stall", 如下图所示:



图 4.4.2 电梯报警键按下后电梯标签的改变

电梯报警之后,如果该电梯之前有被分配外部任务(即按下电梯外部按键后分配的任务),系统会自动将该外部任务分配给最适合的另外一部电梯,如下图所示,当第 1 号电梯报警之后它的外部任务分配给了第 2 号电梯。



图 4.4.3 电梯报警键按下后外部任务的重新分配

## 4.5 电梯介绍文本

电梯介绍文本是使用 QTextBrower 编辑的，是不可编辑的信息。其作用是给用户展示电梯调度系统的基本情况和指南。

2253744林觉凯的电梯调度小程序：  
左侧按下电梯内部按键，右侧按下电梯外部按键；  
同时可以观察电梯运行状态和电梯运行时的交互信息；  
你可以在需要的时候按下左侧每一台电梯下方的开门键、关门键和报警键。  
电梯的交互信息如下：

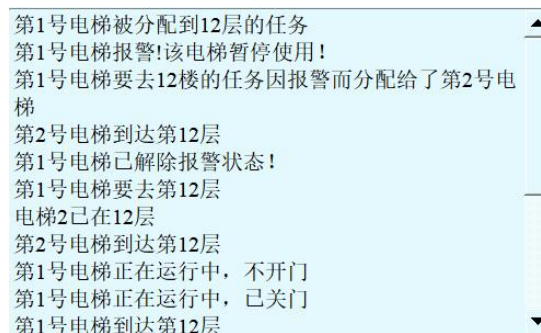
图 4.5 电梯介绍文本

## 4.6 电梯信息文本

电梯交互信息的输出是利用 QTextEdit 编辑的，它可以实时输出电梯的状态、事件信息，每一段电梯信息都是通过如下代码添加到 QTextEdit 中并且显示的：

```
updateInfo = MyWindow.findChild(QTextEdit, 'information')
if updateInfo is not None:
    updateInfo.append(new info)
print(new info)
```

在本项目的用户显示界面中，设置的 QTextEdit 控件的名字为 'information'，我们首先使用 findChild 函数找到该控件，if updateInfo is not None: 这行代码可以确保代码的健壮性，再将 new info 加入文本框中 (new info 为新的电梯交互信息)。



第1号电梯被分配到12层的任务  
第1号电梯报警!该电梯暂停使用!  
第1号电梯要去12楼的任务因报警而分配给了第2号电梯  
第2号电梯到达第12层  
第1号电梯已解除报警状态!  
第1号电梯要去第12层  
电梯2已在12层  
第2号电梯到达第12层  
第1号电梯正在运行中，不开门  
第1号电梯正在运行中，已关门  
第1号电梯到达第12层

图 4.6 电梯交互信息输出文本框

## 4.7 电梯数码管显示和标签

电梯标签有五种状态：上行箭头、下行箭头、Stay、Open 和 Stall，分别表示当前电梯处于上升、下降、关门停留、开门和报警暂停使用状态。电梯的数码管显示通过实时更新的电梯所在层数，来改变相应数码管的整数显示，以此达到电

梯运行楼层的模拟演示。

电梯标签和数码管显示的设置代码如下：

```
#设置电梯数码管的颜色格式
elevatorLCD.setStyleSheet("QLCDNumber{background-color: black; color: #93D5DC;}")
# 设置电梯数码管的显示位数
elevatorLCD.setDigitCount(2)
#设置电梯数码管的数字样式为 Filled
elevatorLCD.setSegmentStyle(QtWidgets.QLCDNumber.Filled)
#设置电梯标签字体格式
elevatorState.setFont(QtGui.QFont("Times new roman", 20, 100))
#设置电梯标签颜色格式
elevatorState.setStyleSheet("QLabel{background-color: black;color: #93D5DC;}")
#设置电梯标签位置居中
elevatorState.setAlignment(QtCore.Qt.AlignCenter)
```



图 4.7 电梯不同的数码显示和标签

## 5.项目总结

### 5.1 项目遇到的问题

本项目主要分为两个部分：前端页面和后端算法。前端页面方面主要是要安排好每一个控件的位置这对于计算长度、宽度有着一定的数学要求；同时每一种控件的格式：比如颜色，字体和不同状态也有相应的考究，需要我们去查找相应的字体格式和颜色的代码表示。后端算法主要是电梯调度的主算法，需要结合电梯的各种情况进行考虑。在编写代码的过程中，由于这是我第一次用 python 编写大项目，所以对 python 的缩进的问题有点小忽视，在一个界面 UI 的设置中由于小缩进的问题，没有在准确的位置显示相应内容，导致我找了半天 bug 才发现问题的所在。这次编程 debug 经历也给了我一定的经验和教训。至于后端的算法方面没有遇到太大的问题，参考了网上相关电梯调度的众多算法，结合在一起写出了本项目的总体调度算法。python 的代码简洁很多，有许多方便调用的函数，这让我的算法可以得到更方便、更好的实现。

## 5.2 项目心得与收获

这是操作系统第一次课程项目处理机管理的电梯调度模拟，我学习到了很多知识和技能。首先是 python 代码的编写。之前写项目基本上都是使用 C++ 语言，这次使用 python 编写程序，让我对 python 语言有了更加深入的理解和熟练的掌握。python 语言简洁很多，有许多方便调用的函数，使得编写更加方便；其次就是前端 PyQt5 库的使用，之前的项目很少做 UI，这次电梯调度模拟第一次做出比较精美设计的页面。同时，我对 PyQt5 这一方便好用的前端界面根据进一步熟悉，这使得我又学会了一项实用技能；最后最重要的是通过这次项目设计，我对线程的理解更加深入。本项目建立的五个电梯线程在执行期间使用信号和槽函数与主线程之间进行通信，主线程中的槽函数通常用于响应从其他线程发出的信号，它发送信号至主线程更新相应的前端界面。通过这次电梯调度项目，我对于操作系统调度过程有了进一步理解，初步学习了在特定环境下多线程编程方法，同时对简单的调度算法有了一定的了解，这些都巩固了我在理论课上掌握的知识。