进程管理-电梯调度

1950072 郑柯凡

指导老师：张惠娟老师

目录

1. 项目背景

某一栋楼共有 20 层，有 5 部互相连接的电梯。每个电梯都有一些按键，如：数字楼层键、开门键、关门键、报警键。同时，在每个楼层，5 部电梯拥有共享的上行键和下行键，即在某一层按下上行键后，5 部电梯的上行键都会亮起。另外，每个楼层每部电梯都有一个独立的数码显示屏显示电梯当前所在的楼层。同时，本项目还有如下假设：

1.所有电梯初始状态都在第一层。

2.每个电梯在没有相应任务的情况下停留在当前楼层。

1.1 需求分析

本项目应满足如下需求：

1.对于电梯内部的用户，按下电梯内部的数字按键后，数字将会高亮显示给予用户提示，同时电梯将会将用户送达目标楼层。

2.对于电梯外部的用户，按下楼层中的上行键或下行键后，系统将自动为用户分配电梯资源，尽可能快地前往用户楼层搭载用户。

3.在电梯内部按下警报键后，整部电梯将会进入维修状态，数码显示屏显示红色高亮提示且整部电梯的内部按键将会失效。

2. 开发环境

开发环境：Windows 10
开发软件：Qt Creator 4.11.0 + Qt5.9.9
开发语言：C++

3. 算法设计

3.1 LOOK 算法

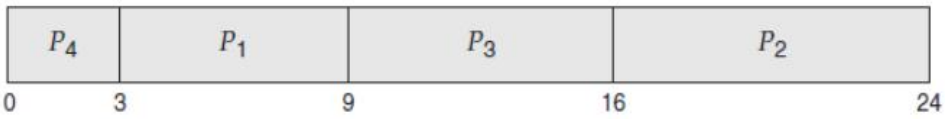
LOOK 算法是 SCAN 算法的一种改进，其核心思想也是扫描。SCAN 算法的基本工作原理是维护一个任务请求表，然后对所有设备从上至下进行循环扫描，若有任务请求则执行，然后继续之前的循环扫描。LOOK 算法的改进就是在每次扫描时不需要绝对地从上至下，从第一个设备到最后一个设备，而是在当前方向的最后一个任务请求处调转方向，这样的好处是不需要扫描一些没有意义的设备，降低算法的时间复杂度。

在本项目电梯调度中，每一个电梯实例都会维护一个楼层请求表，该表是由该电梯内部按键的请求和电梯外部所分配的请求组合而成的。以电梯上行为例，当电梯向上运行时，该电梯会检查当前的楼层请求表，并将同一方向的最后一个楼层请求作为终点。

在到达终点之后，有两种情况。一，该电梯任务请求表为空，即没有其他的任务了，此时令该电梯停留在当前楼层，并将其状态置为空闲。二，该电梯还有其他任务，由于该电梯已处理完方向向上的最后一个楼层，因此剩下的任务只可能是向下的任务请求或方向向上但楼层较低的任务。此时就可以令该电梯转换方向，去处理另一方向的任务。并不断地重复上述的操作。

3.2 非抢占式最短剩余时间优先调度（SJF）

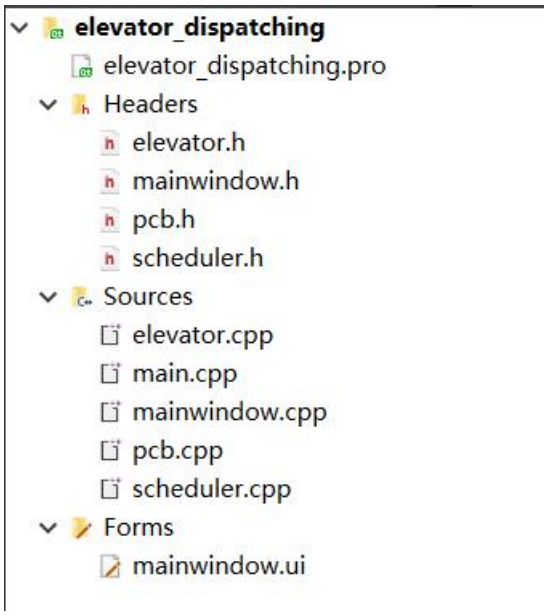
最短剩余时间优先调度算法会优先选择具有最短 CPU 执行时间的进程，如果两个进程具有相同的 CPU 执行时间，则再根据先到先服务的思想进行处理。举个例子，假如有四个进程 P1、P2、P3、P4,CPU 执行时间分别是 6、8、7、3，这四个进程在 SJF 算法下的执行顺序如下图所示。



在本项目中，主要使用最短剩余时间优先调度算法处理电梯的外部请求。当一些电梯的外部请求不符合当前正在运行电梯的方向要求时，比如说电梯正在下行，而某一楼层需要上行，就需要调用空闲的电梯资源，否则会造成用户的等待时间长，即进程的等待时间长。因此，我们采用最短剩余时间优先调度算法，调用离请求楼层距离最近的电梯资源。同时，我们采用的是非抢占式的最短剩余时间优先调度算法，即进程之间不能互相抢占，否则就会出现电梯运行到一半又去处理其他进程的情况，这是不符常理的也是错误的。

4. 系统设计

4.1 类设计



本项目中的类主要分为两大类，一类用来控制图形化界面的交互逻辑，本项目中是mainwindow类,另一类用来实现电梯调度的核心逻辑,包括elevator类、pcb类和scheduler类。主要对核心逻辑类做介绍。

1. **elevator电梯类：**维护电梯对象的状态，便于调度和信息的更新

```
class  Elevator
{
public:
        //进程状态
        int  flag;
        //内部按键
        int  in[20];
        //外部按键
        int  out[20];
        //当前方向的终点
        int  end;
        //当前楼层
        int  nowfloor;
        //运行方向(1代表向上，-1代表向下，0代表停留)
        int  direction;
        Elevator()
        {
                flag=0;
                direction=0;
                end=0;
                nowfloor=1;
```

```
                memset(in,0,sizeof(in));
                memset(out,0,sizeof(out));
        };
        ~Elevator(){};
        //返回电梯状态
        int  getflag(){return  this->flag;}
        //返回电梯运行方向
        int  getdirection(){return  this->direction;}
        //返回当前楼层
        int  getnowfloor(){return  this->nowfloor;}
        //更新电梯工作状态
        void  updateflag();
        //更新电梯进程
        void  updatein(int  x);
        void  updateout(int  x);
        void  updatedirection(int  x);
        void  updateend();
        //电梯运行（处理线程）(每隔 1 秒调用一次)
        bool  run();
};
```

2. **pcb 外部任务进程类：**维护电梯外部按键的任务请求，便于调度器调度

```
class  PCB
{
private:
        int  start;
        int  direction;
        //进程状态，用于分配资源时使用
        bool  status;
public:
        PCB();
        PCB(int  x,int  d)
        {
                start=x;
                direction=d;
                status=0;
        }
        ~PCB(){};
        //返回楼层
        int  getstart(){return  this->start;}
        //返回方向
        int  getdirection(){return  this->direction;}
        //返回状态
        bool  getstatus(){return  this->status;}
```

```
        //修改状态
        void  changestatus(){this->status=1;}
};
```

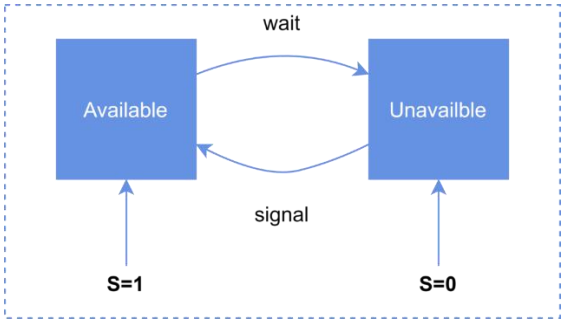
3. **scheduler 调度器类：**使用调度算法为进程分配电梯资源

```
//调度器
class  Scheduler
{
public:
        Elevator  schedule[5];
        queue<PCB>all;
        Scheduler(){};
        ~Scheduler(){};
        //对现有进程进行分配(每隔 1 秒调用一次)
        void  dispatch();
        //外部按键亮，创建一个进程
        void  createPCB(int  floor,int  direction);
};
```

其他内容如电梯内部的按键任务使用信号槽连接（具体原因见 4.2），直接更新电梯类中的内部按键数组，更加方便，就不再另外增添新类。同时，本项目利用信号槽机制，每隔 1000ms 更新一次电梯状态及相应的可视化界面。

```
//连接槽函数和时间函数
QTimer  *timer  =  new  QTimer(this);
connect(timer,  SIGNAL(timeout()),  this,  SLOT(run()));
timer->start(1000);
```

4.2 信号量机制



在本项目中，内部按键请求的处理是相对方便的，只需要修改相应电梯的状态并依照 LOOK 算法执行即可，无需考虑电梯资源分配的问题。相比之下，电梯外部的请求就复杂许多，需要考虑 5 部电梯的运行与否、运行方向等属性，因此我们采用信号量机制进行处理。对于一个外部请求，若以下两个条件满足其中一个，则分配相应的电梯资源给它；若都不满足，则该请求返回就绪队列等待下次调度。

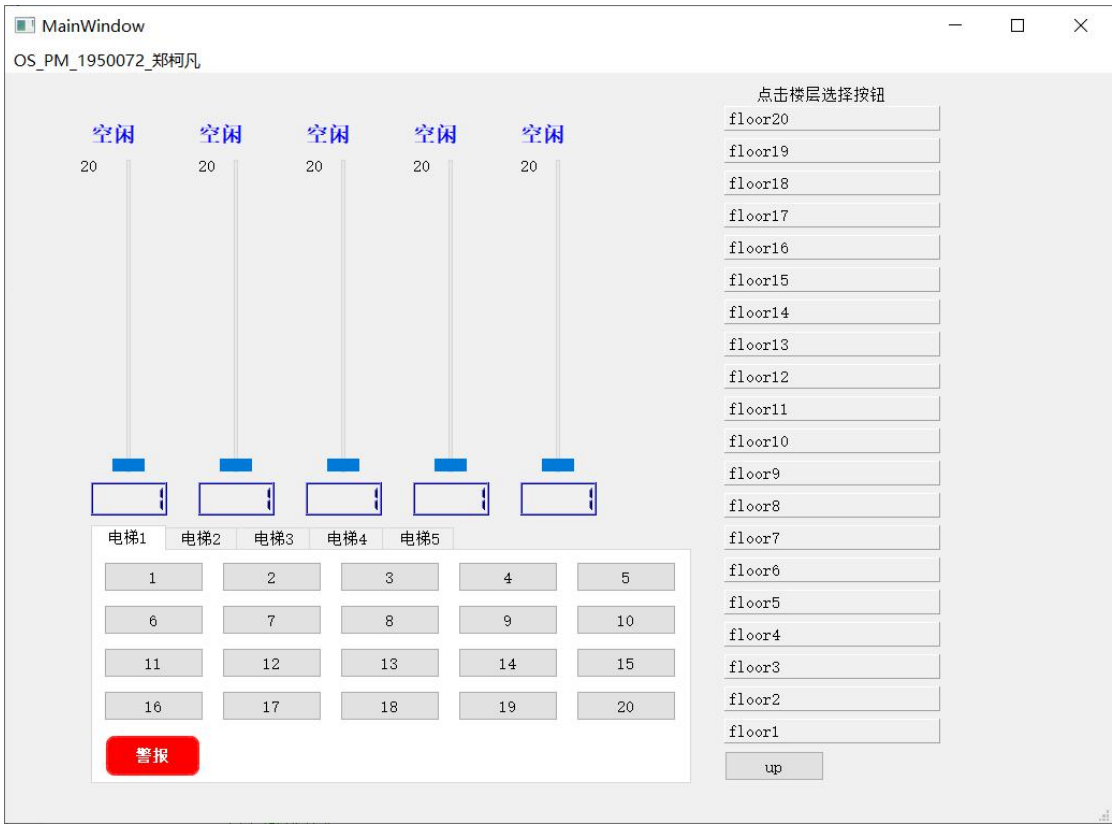
- 有正在运行的电梯，方向相同且楼层没有错过（方向向上则请求楼层在电梯楼层之上，

方向向下则请求楼层在电梯楼层之下）。

- 正在运行的电梯都不合适，但有电梯处于空闲状态。

5. 项目演示

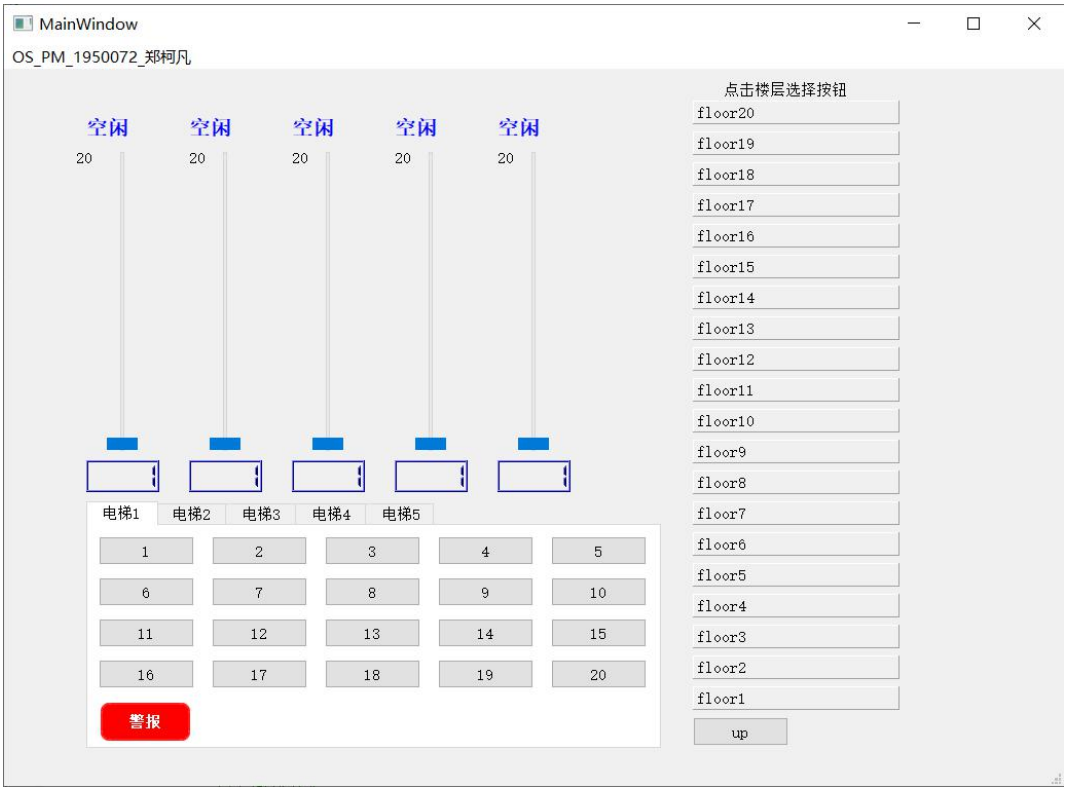
5.1 界面展示



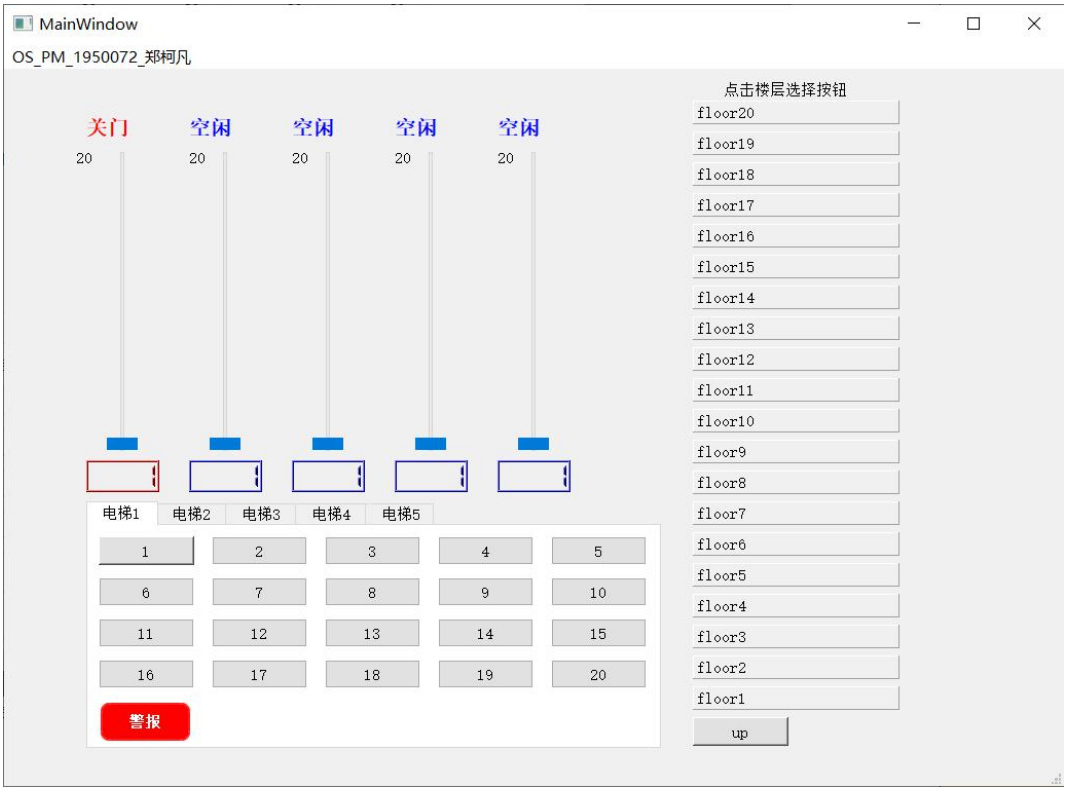
1. **电梯模型：**Qslider 实现。模拟电梯运行时的状态，滑块会根据电梯的楼层滑至相应的位置。
2. **电梯运行状态显示屏：**QLabel 实现。展示电梯的状态，有“空闲”、“运行”、“开门”、“关门”、“维修”五种状态。
3. **电梯楼层显示屏：**QLCDNumber 实现。实时展示电梯所处的楼层情况。
4. **电梯内部楼层按键及警报键：**QtabWidget 窗口和 QPushButton 实现。电梯内部的按键可供用户选择点击，点击之后会呈现红色高亮显示，提示用户系统已接受请求，同时会执行相应的任务。警报键也可供用户点击，点击后将提示其他的用户该电梯无法运行需要维修，同时该部电梯也不会响应电梯内外部的各种请求，用户的请求将全部由其他电梯承担。
5. **电梯外部各楼层按键：**QToolBox 窗口和 QPushButton 实现。用户可以点击相应的楼层按键展现该楼层的上行键和下行键，再次点击上行键或下行键后按钮将呈现红色高亮显示，提示用户请求已被电梯系统接受，同时为其分配相应的电梯资源。

5.2 操作说明

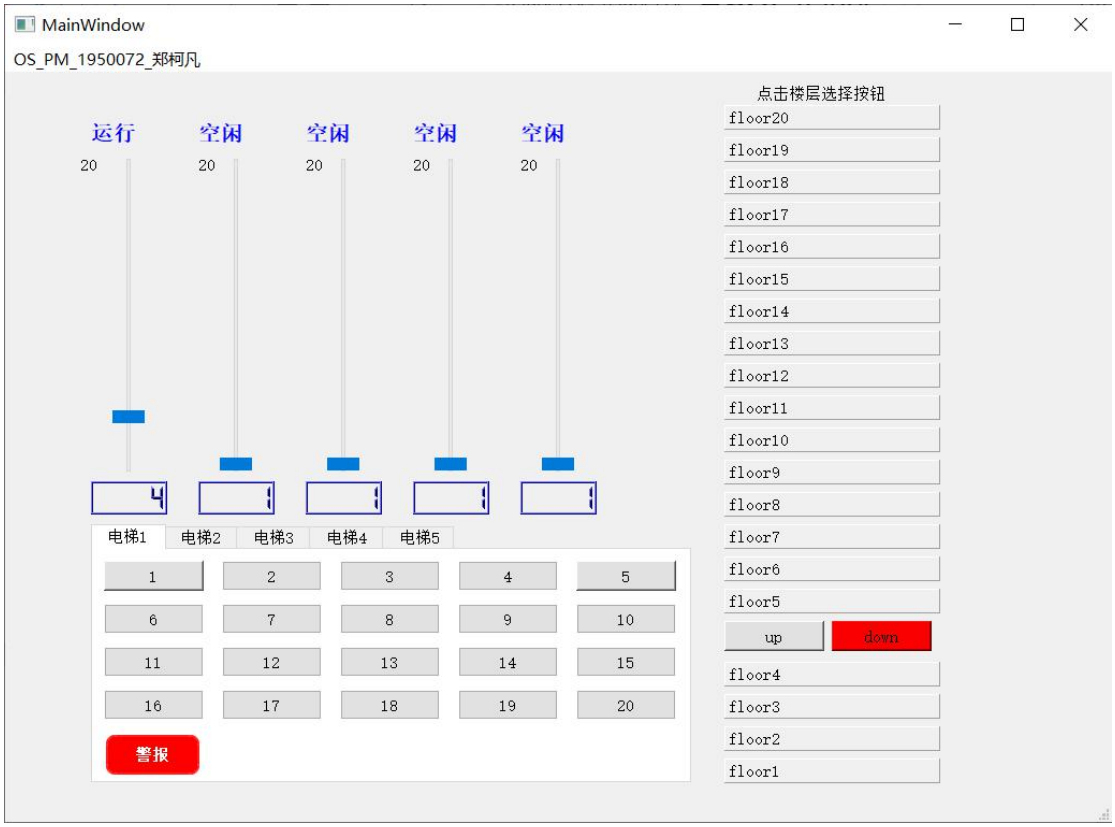
- 以管理员权限运行*elevator_dispatching.exe*，进入电梯模拟系统。



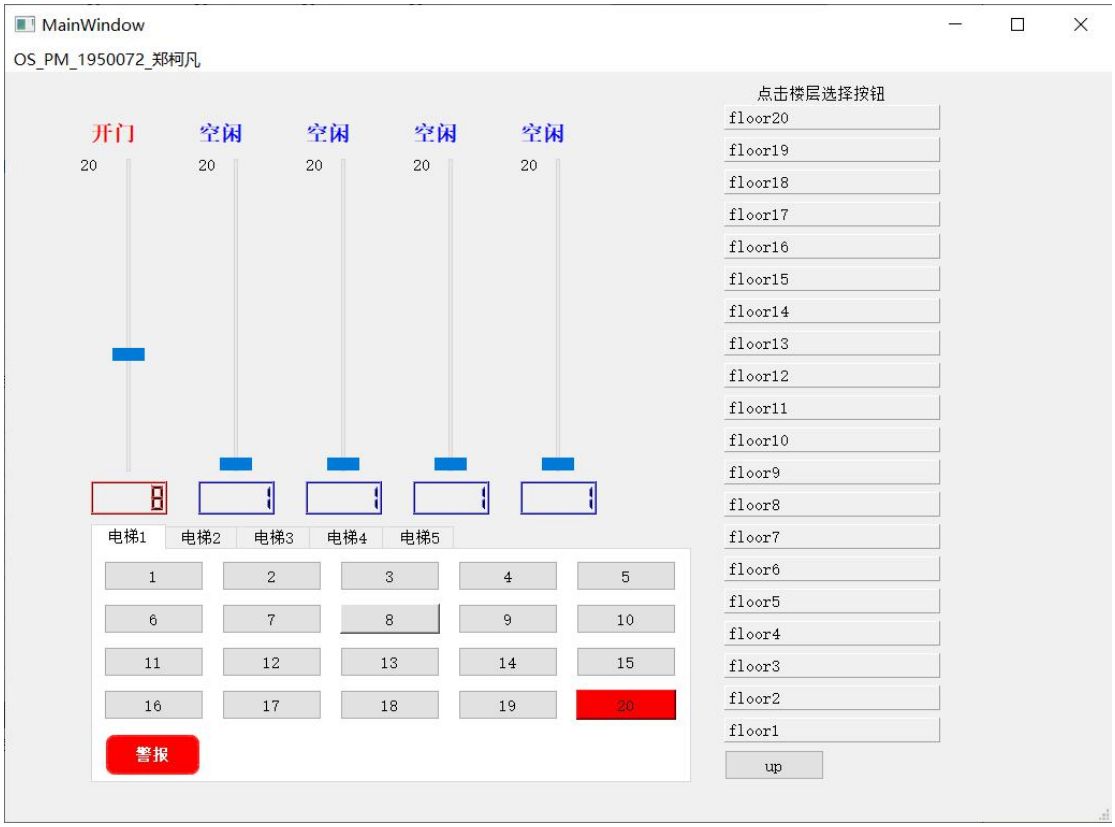
- 点击电梯外部楼层按钮，电梯将响应请求运行至该楼层并展示开门、关门状态。有电梯在相应楼层（图中按下 floor1 的上行键）



无电梯在相应楼层，电梯会前往相应楼层（图中按下 floor5 的下行键）

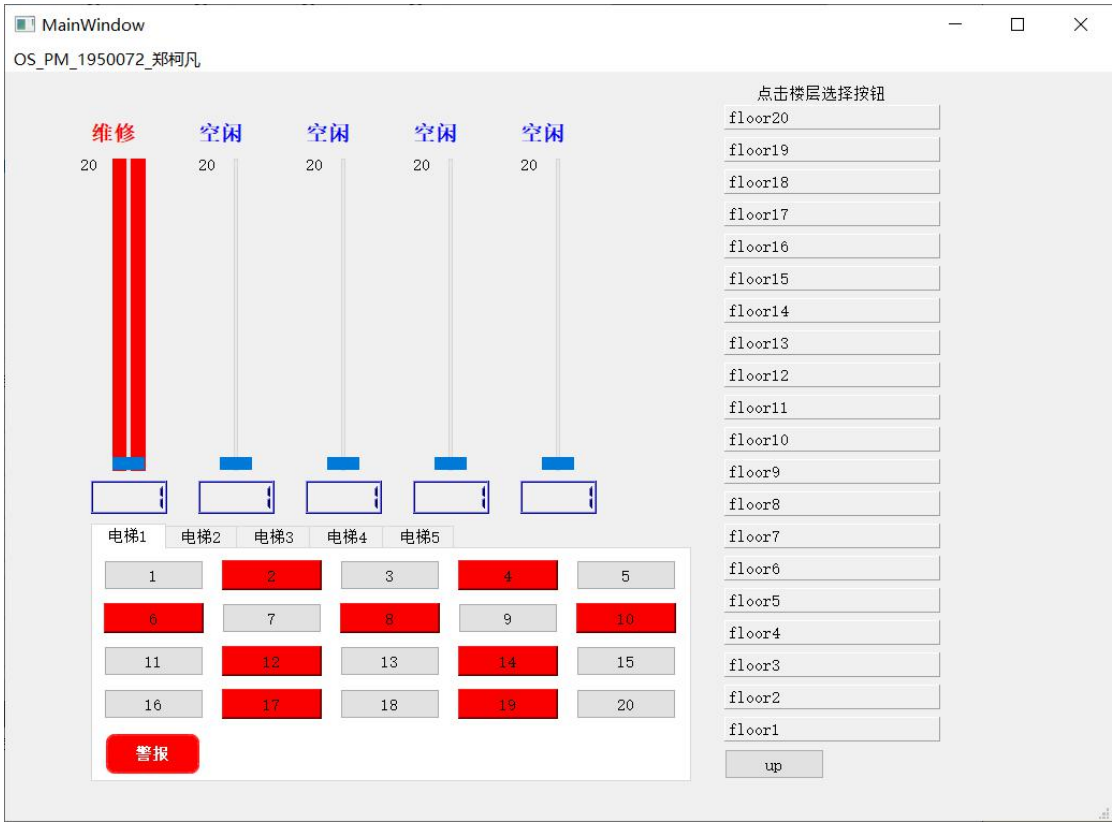


- 点击楼层内部按钮，若电梯就在该楼层，则电梯直接进入开门、关门状态；若电梯不在该楼层，将前往该楼层。如果电梯在运行过程中有其他请求，将按照 LOOK 算法执行。（图中按下电梯 1 的 8 楼和 20 楼，电梯依次处理）



- 点击楼层内部警报键，电梯将全部红色高亮提醒用户，并在状态显示屏中显示维修。同

时该电梯内部全部按键失效。（图中按下电梯 1 的警报键）



5.3 多线程运行展示