基于线程思想的电梯调度设计方案报告

学号：2252063 姓名：闵孟宇

【开发环境】

Visual Studio 2022+Qt6.5.3+C++

【本项目完成度】

1.（完成）每个电梯里面设置必要功能键：如数字键、关门键、开门键、上行键、下行键、报警键、当前电梯的楼层数、上升及下降状态等。2.（完成）每层楼的每部电梯门口，应该有上行和下行按钮和当前电梯状态的数码显示器。

3.（完成,因为五部电梯共享一个控制台）五部电梯门口的按钮是互联结的，即当一个电梯按钮按下去时，其他电梯的相应按钮也就同时点亮，表示也按下去了

4.（完成）所有电梯初始状态都在第一层。每个电梯如果在它的上层或者下层没有相应请求情况下，则应该在原地保持不动。

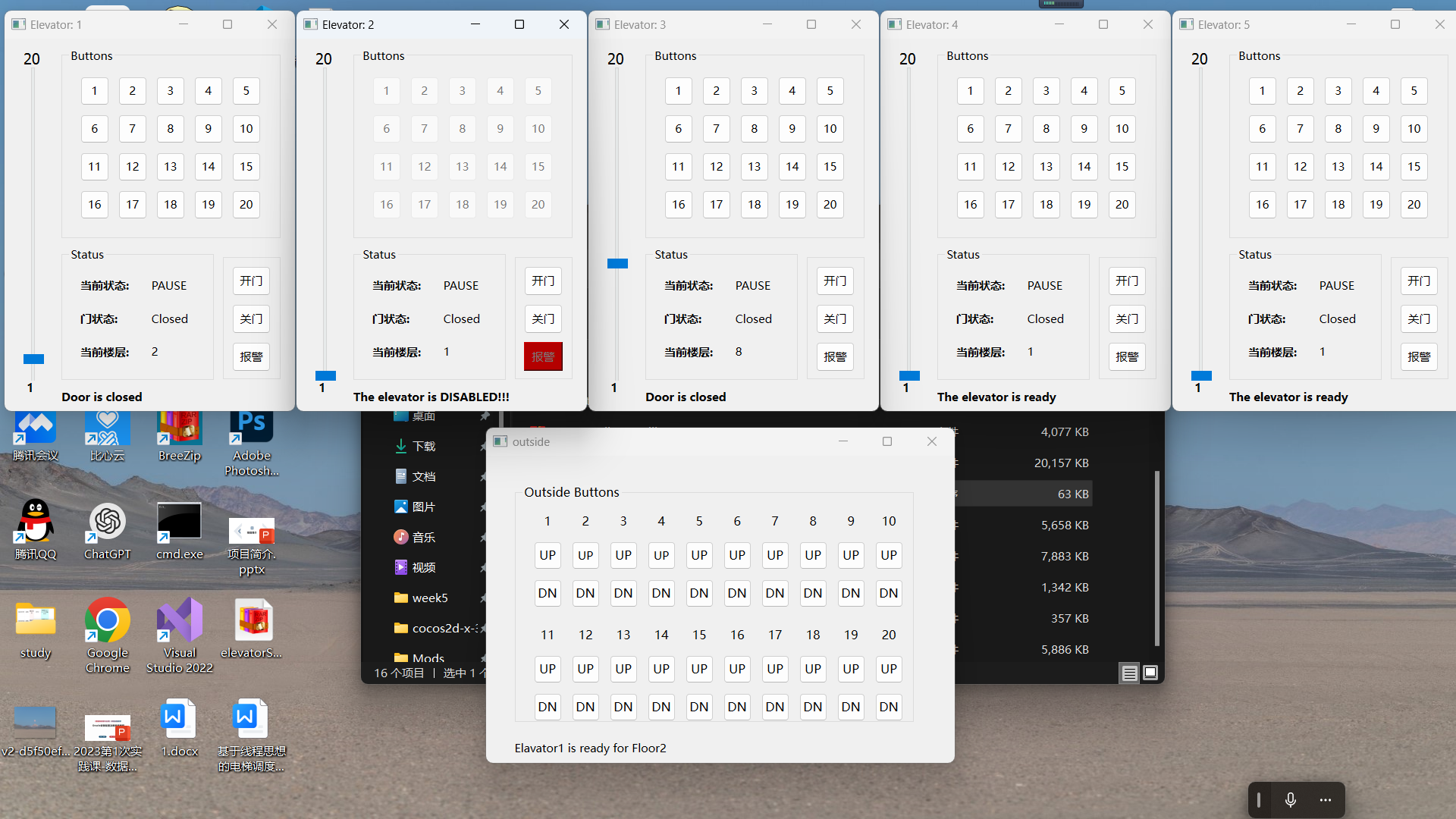
5.（完成）多线程思想，每部电梯都是独立的线程，独立运行互不干扰。

6.（完成）在每个ui界面底部都有提示信息。比如电梯有没有损坏，此时人正在进出，以及把哪部电梯分配给了哪个楼层，等等。

【运行方法】

解压elevatorScheduling.zip，双击其中Os1.exe

【运行示例】



【类的设计】

1. 电梯类Os1

该类主要完成单个电梯内部ui界面的设置、内部调度算法的实现（将电梯内部任务统筹管理，生成最佳任务列表）以及电梯状态的更新。

**涉及到的关键变量包括：**

int status = PAUSE; //电梯状态

int currentFloor = 0; //电梯当前楼层

int is\_available = YES;//电梯是否可用

vector<int>dest\_inside; //电梯当前的顺路的任务队列

vector<int>dest\_inside\_no; //电梯当前的顺路的任务队列

**涉及到的主要函数包括：**

//单个电梯内部ui界面初始化

Os1(QWidget \*parent = nullptr, int elevator\_id = 0, int floor\_num = 20);

//把电梯内乘客选择的目的地调度到电梯的任务列表中

void task\_inside(int elevator\_id, int destnation\_id);

//电梯运行过程

void to\_run();

//门的打开与关闭，持续两秒，模拟乘客进出

void open\_door();

//电梯状态更新

void refresh();

1. 外部选择类outside

该类主要实现外部选择ui界面的初始化、电梯外部乘客的呼叫选择（在哪一楼层想要上或者下）、外部调度算法的实现（为某楼层的乘客分配最佳电梯）。

涉及到的关键变量包括：

vector<Os1\*> elevators;//电梯列表

vector<int>enabled\_elevators;//可用电梯列表序号

vector<pair<int, int>> ele\_priority;//每部电梯的优先度

涉及到的主要函数有：

//电梯外部选择ui界面的实现

outside(QWidget\* parent = nullptr, vector<Os1\*> Elevators = {},int floor\_num = 20);

//电梯外部调度时优先度的计算

int priority\_cal(int user\_choice,int user\_floor,int ele\_status,int ele\_floor, int last\_task\_floor);

//电梯外部调度，将任务添加到对应电梯内部任务列表中

void select\_outside(int choice, int floor);

1. 主要界面类all\_in\_out

该类主要是为了实例化ELEVATOR\_NUM个电梯，以及由这ELEVATOR\_NUM个电梯共享的电梯的外部选择（控制台）。

【主要算法】

1. 内部调度算法

对于电梯本身而言，其内部的任务逻辑是非常简单且不受外都调度干扰的。

在本算法中，电梯的任务根据电梯此时的运行状态分成了顺路的任务列表dest\_inside以及不顺路的任务列表dest\_inside\_no。

比如当电梯此时在5楼且运行向上，存在终点为7、3、8、6、5、1层楼；则顺路的任务包括7、8、6、5；不顺路的任务包括3、1；然而dest\_inside以及dest\_inside\_no并非简单的将任务添加进去，还包括对任务优先度的排序。优先度根据电梯会最先到达哪个楼层而定。所以此时dest\_inside为5、6、7、8；而不顺路为3、1。

同理当电梯此时在5楼且运行向下，则上面例子中dest\_inside为5、3、1；而不顺路的任务包括6、7、8。

电梯实际运行的任务只有dest\_inside；当dest\_inside为空且dest\_inside\_no不为空时，则不顺路任务列表变为顺路任务列表，且原不顺路列表清空，电梯将继续反向运行。

直到两者均为空时，电梯才真正PAUSE空闲下来。

task\_inside(int elevator\_id,int destnation\_id)代码如下，根据电梯内部乘客此时选择的目标楼层与电梯当前楼层的相对位置以及电梯的运行状态进行分类：

if (currentFloor <= destnation\_id){

if (status == PAUSE)

dest\_inside.push\_back(destnation\_id);

else if (status == UP)

添加进dest\_inside且由小到大排序

else//status==DOWN

添加进dest\_inside\_no且由小到大排序

}

else if (currentFloor > destnation\_id){

if (status == PAUSE)

dest\_inside.push\_back(destnation\_id);

else if (status == DOWN)

添加进dest\_inside且由大到小排序 }

else//status==UP{

添加进dest\_inside\_no且由大到小排序 }

}

1. 外部调度算法

外部调度算法用于把最优电梯分配给某一楼层的乘客，其中优先度是非常值得关注的参数。本算法中的优先度实际上算的是楼层的距离。

**设乘客在a层按下了上或下的按钮，某一电梯此时在b层，其电梯内部顺路的任务队列中最后一个任务的目的楼层为c层。则该电梯的优先度计算如下：**

|  |  |  |  |
| --- | --- | --- | --- |
| 电梯status | 优先度计算 | | |
| PAUSE | /a-b/ | | |
| UP | 电梯floor  >乘客floor | 乘客UP | /c-a/+/c-b/ |
| 乘客DOWN | /c-a/+/c-b/ |
| 电梯floor  =乘客floor | 乘客UP | /a-b/ |
| 乘客DOWN | /c-a/+/c-b/ |
| 电梯floor  <乘客floor | 乘客UP | /a-b/ |
| 乘客DOWN | /c-a/+/c-b/ |
| DOWN | 电梯floor  >乘客floor | 乘客UP | /c-a/+/c-b/ |
| 乘客DOWN | /a-b/ |
| 电梯floor  =乘客floor | 乘客UP | /c-a/+/c-b/ |
| 乘客DOWN | /a-b/ |
| 电梯floor  <乘客floor | 乘客UP | /c-a/+/c-b/ |
| 乘客DOWN | /c-a/+/c-b/ |

**经过分类后，可以发现，只有在（1）电梯空闲（2）电梯运行且人请求方向与电梯运行方向一致且人在电梯运行方向上 时，优先度才为/a-b/，其他情况均可以以/c-a/+/c-b/计算。由此较为轻松地写出priority\_cal优先度计算函数**

外部调度过程会对于每一个乘客，向其分配优先度最好的电梯，并将乘客所在楼层作为一个终点，来调用前述task\_inside(int elevator\_id,int destnation\_id)函数将其添加到电梯内部或顺路或不顺路的任务队列中，相当于电梯内部有乘客按了此楼层的按钮。

1. 电梯运行状态的更新。

电梯每上一层或每下一层都算是一次动作，这样的一次动作在本项目中被写入了to\_run()函数。通过Qt中的计时器来设置每0.8s运行一次to\_run（）函数，就可以实现电梯的一直运行。

对于每一次to\_run()，可以根据电梯顺路任务队列dest\_inside和不顺路任务队列dest\_inside\_no的情况进行分类。

当dest\_inside.empty() && dest\_inside\_no.empty()两者皆空，说明电梯此时空闲。

如果dest\_inside不空，则根据电梯当前楼层与顺路任务队列的第一个目标楼层之间的大小比较来确定电梯运行方向，其中当两者相同时说明电梯到达了目的地，需要把它从顺路任务队列中删除。

当dest\_inside空而dest\_inside\_no不空，说明电梯需要反向了，就把不顺路改成顺路，再把原来不顺路删除。

to\_run（）代码如下：

if (dest\_inside.empty() && dest\_inside\_no.empty())

status = PAUSE;

else if (!dest\_inside.empty()){

if (currentFloor < dest\_inside[0]) {//则电梯上升

ui.label\_5->setText("The elevator is up");

status = UP;

currentFloor++;

}

if (currentFloor > dest\_inside[0]){

ui.label\_5->setText("The elevator is down");

status = DOWN;

currentFloor--;

}

if (currentFloor == dest\_inside[0]){

open\_door();//模拟乘客上上下下

while (!dest\_inside.empty()&&currentFloor == dest\_inside[0])//因为在把任务从外调内时可能会楼层重复

dest\_inside.erase(dest\_inside.begin());//删除当前楼层的任务

if (dest\_inside.empty())

status = PAUSE;

buttons[currentFloor]->setEnabled(true);

buttons[currentFloor]->setStyleSheet("");

}

}

else if (dest\_inside.empty() && !dest\_inside\_no.empty()){//一个方向跑到头了

dest\_inside = dest\_inside\_no;

dest\_inside\_no.clear();

}