# 操作系统处理机管理项目一

## 说明文档

1452716

张尹嘉

## 一 项目需求

· 模拟十字路口的交通控制情况。

· 考虑东、西、南、北四个方向，每个路口设置一盏红绿灯，为简单起见，每种灯显示时间为8秒。

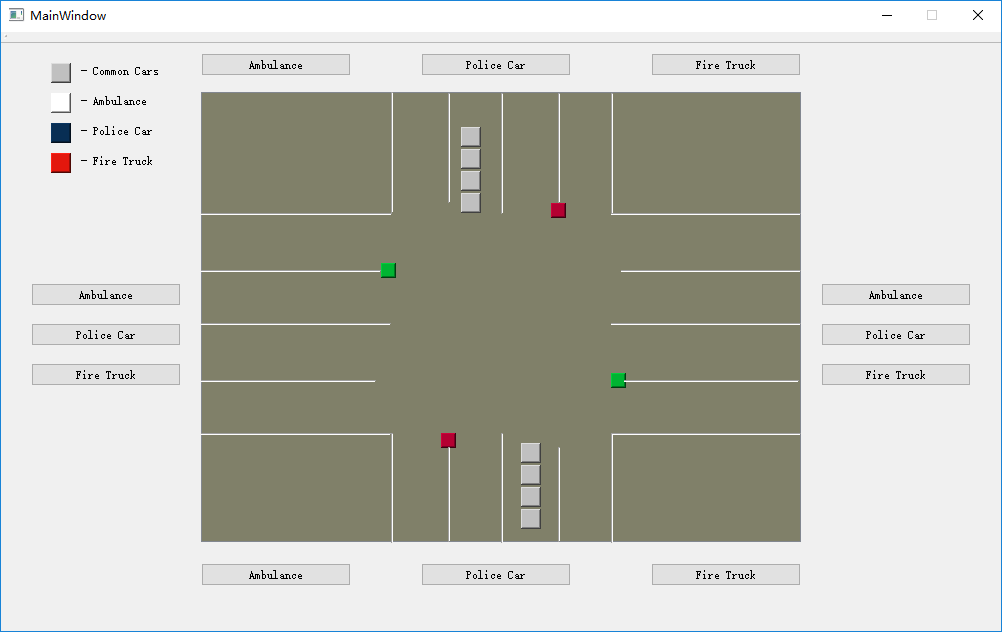
· 当东西（或南北）方向红灯时，所有车辆（除了消防车，救护车，警车）均排队等待，当东西（或南北）方向绿灯时，所有车辆按序行驶。

## 二 程序运行环境及运行样例

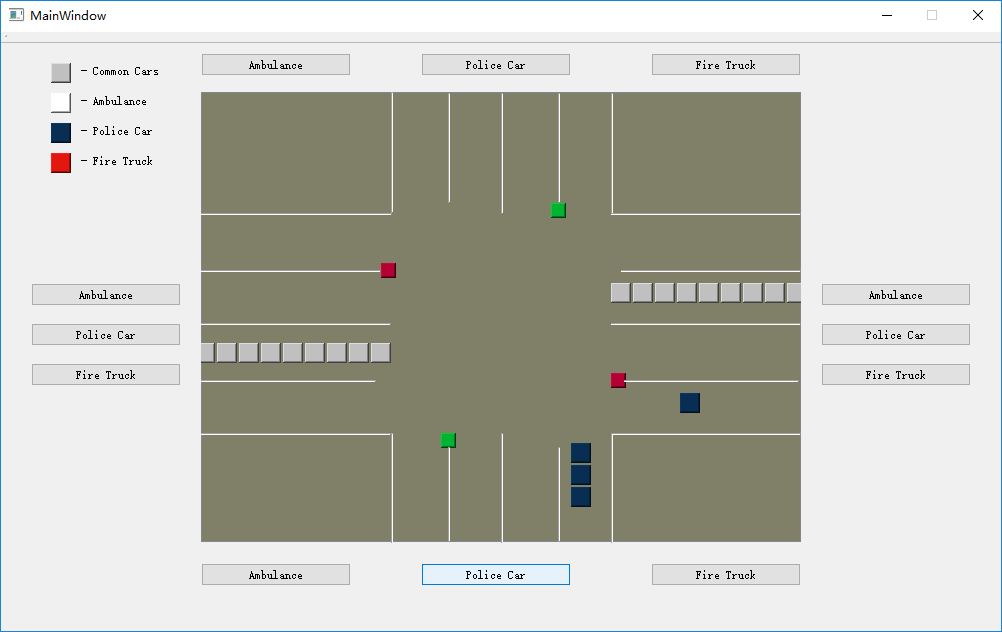
· 本程序基于VC++2015和Qt框架进行开发，ExeFile文件夹内自带了程序所需的链接库文件。点击osProjectOne.exe可直接运行，请在Windows 10上进行运行测试。

· 以下为程序使用样例（左上角为各种车辆的示意图说明）：

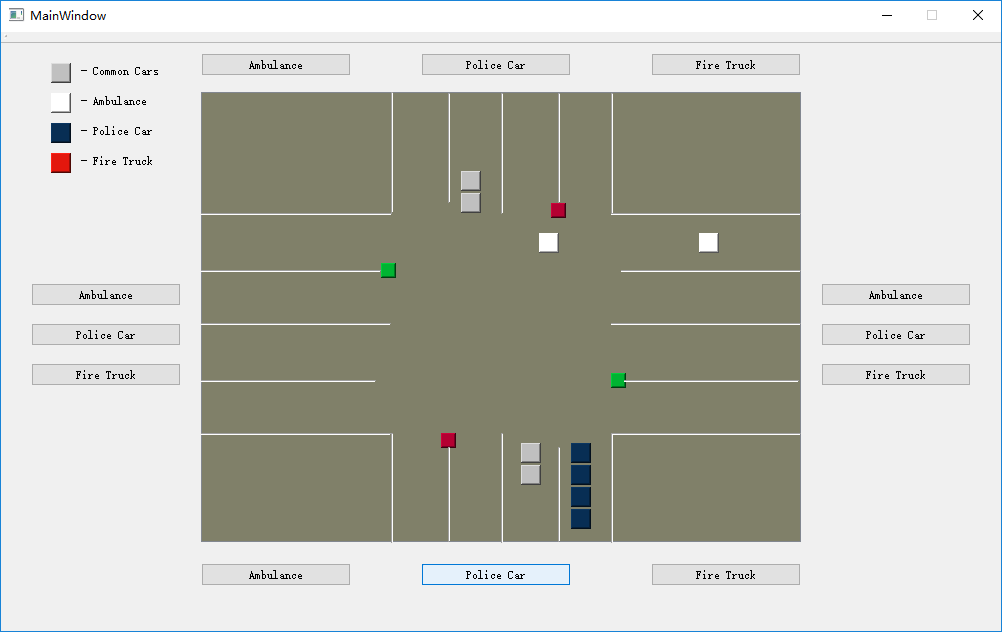
(1) 开始运行时，没有特殊车辆，红绿灯各八秒，两个路口交替运行。



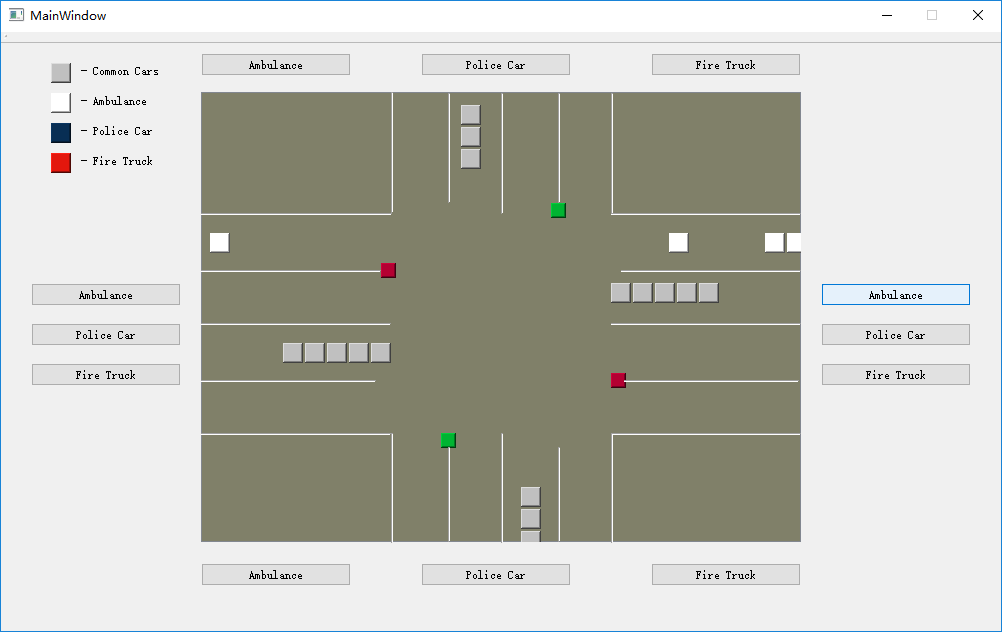
(2) 在四个方向上各有三个按钮，可以添加救护车，警车和消防车，特种车辆占用专门车道。



当该道路时红灯时，所有车辆按序行驶。



当非行驶中的道路上有特种车辆来时，特种车辆优先行驶，此时绿灯的道路会暂停行驶。



## 三 实现算法

本程序主要使用了操作系统中的进程同步、互斥，以及多级队列的原理。算法上把车辆当作最小的进程，东西向的普通车辆，南北向的普通车辆，自西向东的特种车辆，自东向西的特种车辆，自南向北的特种车辆和自北向南的特种车辆作为6个进程队列，6个进程同时开始执行，根据全局变量的改变做出相应的反应。进程队列内部按照FCFS的算法进行调度。同方向的普通车流（进程队列）和特种车流可以共享路口资源，而不同方向的车流间是互斥的，采用PV对进行调度，而且特种车辆可以抢占路口资源。两个不同方向的特种车流间是不可抢占的，服从FCFS。

本算法设置三个信号量，crossRoad、emergency和runningEmergency初始值分别是1，1，-1。crossRoad 控制普通车流对路口资源的互斥，emergency控制不同方向车流之间的互斥， runningEmergency 控制同方向的特种车流的同步。具体算法伪代码如下：

(1)普通车流（代码中见ThreadManagement::runOfCommonCars()函数 ）：

forever{

//P操作

crossRoad --;

if crossRoad 小于 0

进程阻塞；

//访问互斥资源

while(对应路口的交通灯是绿灯)

{

车辆依次通过；

while 路口被特种车辆抢占

等待一小段时间；

}

//V操作

crossRoad++;

if crossRoad <= 0

唤醒另一条路口的车流；

//进行下一次循环，重新等待资源

}

(2)特殊车流

forever{

while 该方向上没有特种车辆

等待0.2秒；

//P操作

emergency--;

if emergency < 0 且 runningEmergency%2 ！= 车流编号%2

//说明：当emergency小于0时说明有特种车辆在占用路口资源，但是如果正在用路口的车辆和该进程的方向不冲突，则不用阻塞进程。判断方向是否冲突的方法如下：设东西向的特殊车流分别编号为3和5，南北向编号为2和4。runningEmergency在东西方向第一次有特殊车辆时，由-1变成3，第二次对面方向来车时，会自加2变为5；而南北方向第一次由-1变为2，第二次自加2变为4。当判断当前运行的特殊车辆是否与该进程冲突时，只要runningEmergency和车流编号同时 mod 2 就可以判断了。

进程阻塞；

if runningEmergency != -1

runningEmergency = 车流编号%2 +2； //该方向第一次来车

else

runningEmergency += 2; //该方向第二次来车

//访问互斥资源

屏蔽信号灯；

while 车流不为空

车辆按序行驶；

开放信号灯；

//V操作

emergency++;

if runningEmergency >= 4 //该车流不是此方向的最后的车流

runningEmergency-=2;

else //此方向的最后一个车流

runningEmergency = -1;

if emergency <= 0

唤醒另一个等待的车流；

}

## 四 代码实现

代码共有四个文件，main.cpp是程序主入口，ui\_mainwindow.cpp是程序界面文件，MainWindow.h和MainWindow.cpp是程序逻辑的实现文件。

(1) 自定义数据类型：

车的类型：

enum kind{

CommonCar,

Ambulance,

PoliceCar,

FireTruck

};

车辆方向：

enum direction

{

TopToBottom,

BottomToTop,

LeftToRight,

RightToLeft

};

车流进程队列上的节点：

struct ThreadQueue

{

Car\* thread;

ThreadQueue \*nextThread;

};

(2) 公共变量：

控制普通车互斥的信号量

static int crossRoad = 1;

控制特种车辆互斥和同步的信号量

static int emergency = 1;

static int runningEmergency = -1;

交通灯变量

static int tbLight = 1;

static int rlLight = 0;

控制车辆和交通灯的定时器

static QTimer timerForCar;

static QTimer timerForLight;

6个方向的车流队列

[0]-top/bottom, [1]-left/right,

[2]-top to bottom special cars, [3]-left to right special cars

[4]-bottom to top special cars, [5]-right to left special cars

static ThreadQueue\* SourceDir[6] = { nullptr, nullptr, nullptr, nullptr, nullptr,nullptr };

Qt功能变量，控制线程等待

static QWaitCondition\* waitThread = new QWaitCondition;

static QWaitCondition\* waitThread2 = new QWaitCondition;

Qt功能变量，控制改变公共变量的锁定和解锁

static QMutex\* mutex = new QMutex;

static QMutex\* mutex2 = new QMutex;

(3)

实现中共有三个类，MainWindow类， Car类和 ThreadManagement类。

MainWindows类主要控制界面布局，信号灯，创建普通车辆，通过按钮加入特种车辆，运行动画以及调用ThreadManagement类的调度入口。

Car类负责定义车辆的图标和主要行为。

ThreadManagement类控制进程的运行和调度，是算法的主要逻辑。

MainWindow类改变信号灯等全局环境。ThreadManagement的六个对象根据全局环境和自己的运行逻辑对变化做出反应。六个进程并行执行。

|  |  |  |  |
| --- | --- | --- | --- |
| MainWindow类，继承自QMainWindw | | | |
| 权限 | 类型 | 名称 | 作用 |
| public |  | MainWindow(QWidget \*parent = 0); | 构造函数，将按钮的信号函数和对应的槽函数连接 |
| public |  | ~MainWindow(); | 析构函数 |
| public | void | runAttemper(); | 运行受保护的调度器方法 |
| public | void | changeLights(); | 改变交通灯的颜色 |
| public | void | runAnimation(Car\* car); | 槽函数，运行车辆的动画 |
| public | void | moveCars(int source, int sumcars); | 槽函数，将绿灯结束后还未通行的车辆移动到路口 |
| protected | void | ButtonClicked(); | 槽函数，响应界面上按钮被按下的动作，加入特种车辆 |
| protected | void | createThreadQueue(); | 槽函数，向每个普通车流加入车辆 |
| protected | void | runProtectedAttemper(); | 初始化进程管理队列，初始化交通灯，连接定时器信号和加入车辆的槽函数，连接定时器信号和改变交通灯的槽函数，运行ThreadManagement的调度方法。 |
| private | MainWindow\* | ui | 界面组件的总指针，具体组件在ui\_MainWindow.cpp文件中 |
| private | ThreadManagement \* | mThreadManagerOne; | 南北向的普通车流管理类指针 |
| private | ThreadManagement \* | mThreadManagerTwo; | 东西向的普通车流管理类指针 |
| private | ThreadManagement \* | mSpecialCarsTB; | 自北向南特殊车辆管理类指针 |
| private | ThreadManagement \* | mSpecialCarsBT; | 自南向北特殊车辆管理类指针 |
| private | ThreadManagement \* | mSpecialCarsLR; | 自西向东特殊车辆管理类指针 |
| private | ThreadManagement \* | mSpecialCarsRL; | 自东向西特殊车辆管理类指针 |

|  |  |  |  |
| --- | --- | --- | --- |
| Car类，继承自QObject | | | |
| 权限 | 类型 | 名称 | 作用 |
| public |  | Car(direction Dir, kind Kind, QObject\* parent); | 构造函数，初始化车辆的方向和种类以及父母 |
| public |  | ~Car(); | 析构函数 |
| public | direction | getDir(); | 获取车辆方向 |
| public | kind | getKind(); | 获取车辆种类 |
| public | void | run(); | 车辆运行 |
| public | void | sendSignalToRunAnimation(Car\* car); | 信号函数，发信号给MainWindow类的动画运行函数 |
| private | direction | mDir; | 私有成员变量，车辆的方向 |
| private | kind | mKind; | 私有成员变量，车辆的种类 |
| public | QPushButton\* | CarButton; | 共有成员变量，车辆在界面上的图标 |

|  |  |  |  |
| --- | --- | --- | --- |
| ThreadManagement类,继承自QThread类 | | | |
| 权限 | 类型 | 名称 | 作用 |
| public |  | ThreadManagement(int source); | 构造函数，初始化方向，初始化车流中车辆总数为0 |
| public | static  void | schedule(  ThreadManagement\* tm1, ThreadManagement\* tm2, ThreadManagement\* tm3, ThreadManagement\* tm4, ThreadManagement\* tm5, ThreadManagement\* tm6); | 运行六个方向的调度函数 |
| public | void | run(); | 运行函数，根据种类分别调用两种运行方法 |
| public | void | addSumCars(int pa); | 向总车数中加pa |
| public | int | getSumCars(); | 获取总车数 |
| public | int | getSource(); | 获取方向 |
| public | void | signalsToMoveCars(int source, int sumcars); | 信号函数，发信号给MainWindow的moveCars()方法 |
| public | void | runOfCommonCars(); | 普通车辆的运行方法 |
| public | void | runOfSpecialCars(); | 特种车辆的运行方法 |
| private | int | mSource; | 成员变量，车的方向 |
| private | int | mSumCars; | 成员变量，车辆总数 |