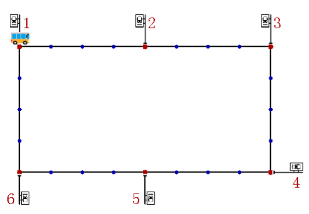
**公交车调度系统**

1. **公交车调度系统问题描述**



环形轨道，一辆车，车辆可以双向任意行驶。我们规定车辆的原始位置为0（该位置也是车站1的位置），按顺时针方向每个单位位置坐标加1。如果轨道总长为10，则按顺时针方向走，位置9的下一个为位置0。车站编号同理，也是按顺时针方向依次递增。车速固定，每秒一个单位。停车接人或乘客下车时需要停车一秒钟。无论一次停站完成几个服务停留时间统一为1秒钟。各站之间距离相等，车辆经过站点时，根据调度策略，车辆可以停也可以不停。其他位置不允许停车。车辆只能在站点停站时才能改变行驶方向。

现在要求我们通过编程实现三种策略下公交车的调度。

1. 先来先服务策略（FCFS）

将所有乘车请求按发出时间排队，然后按队列顺序逐一完成。

1. 最短寻找时间优先算法（SSTF）

这种策略选择下一个服务对象的原则是最短寻找时间，且加入特定的顺便服务，允许途径时停靠顺便服务。

1. 顺便服务策略（SCAN）

这种策略第一个目标是全部已发出请求中完成时间最短的那个请求，第一次行驶方向是完成第一个目标的最短路程的方向。完成服务时，无须考虑顺、逆的请求方向。在行驶过程中，如果所有的请求按照当前的行驶方向找出的最短完成时间都超过跑完轨道一半距离时间时，应该切换行驶方向。

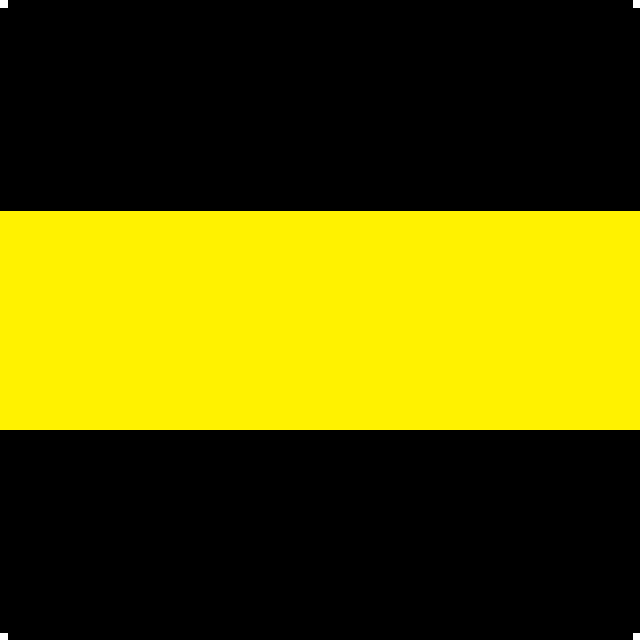
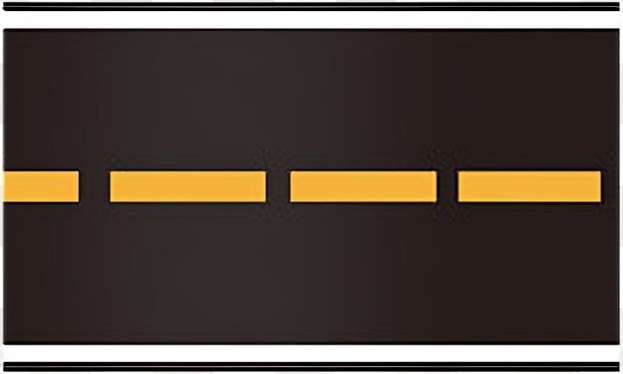
1. **需求分析**

需求分析需要从系统的数据、功能和行为三方面进行分析。

功能方面，主要是模拟出公交车在不断接收指令下，按照相应的策略在已知车站数量的环形轨道上行驶的过程。我们要分析清楚车站数量、指令要求以及所选取的策略；行为方面，需要分析公交车的运行方向和运行时间等等；数据方面，需要分析清楚文件中引导公交车运行的控制指令。

**（1）道路的配置**

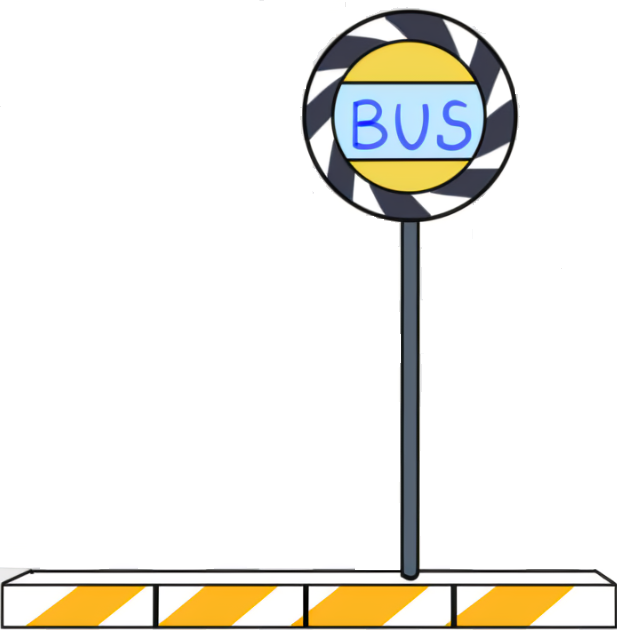
1. 道路的表示

descript

1. 道路基本类型

实际的道路会有单向道路，双向道路；在道路上掉头时需要进入旁侧道路。在此，为了简化问题，不考虑以上情况，只有一条道路，并且可以随时掉头，不花费额外时间。

**（2）车站的配置**



在道路上有此标识的地方视为车站，汽车可以在此处停留完成任务。

**（3）公交车的行驶规则与要求**

1. 车辆的表示。

车辆可以用贴图或者质点表示表示。此处使用贴图。

1. 车辆的运行轨迹。

车辆沿着闭合轨迹顺时针或逆时针行驶，无需掉头或转弯。

1. 车辆和车站的关系。

车辆在目的地车站需要停止一秒钟。

1. 车速的设计要求

车速为恒定的1单位路程每单位时间，到达目的地时立即减为0。

5）车辆的数量

实际中，可能会有较多公交车同时工作，共同处理调度问题，但比较复杂。此处只讨论一个公交车的情况。

**（4）调度指令和调度策略**

1. 调度指令

包含：clockwise n，counterclockwise n，target n，end.

clockwise n表示在车站n有一个顺时针的上车请求；counterclockwise n表示在车站n有一个逆时针的上车请求；target n表示车内有一个在车站n的下车请求。

1. 调度策略

根据文件给出的要求，用指定的策略控制公交车完成调度。

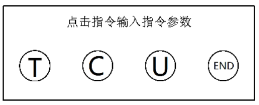
**（5）输入输出**

1. 输入

输入包括配置文件和指令。

配置文件：配置文件通过文件读取，给出公交车的调度策略，车站数，车站的间隔。如果没有给出，则视为默认值。

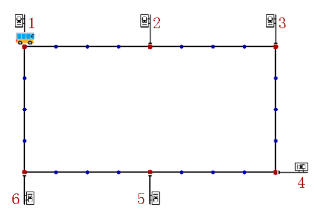
指令：指令可以通过键盘输入或者在图形化界面的输入区输入。指令包括：clockwise n，counterclockwise n，target n，end.通过键盘一行输入一个指令或者在图形化界面点击按钮输入指令。通过输入指令，模拟公交车遇到的情景，从而体现公交车的调度策略。



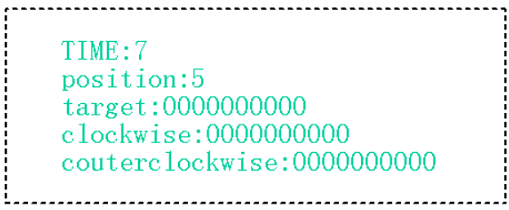
1. 输出

每n秒视为一个单位时间，刷新公交车的状态。

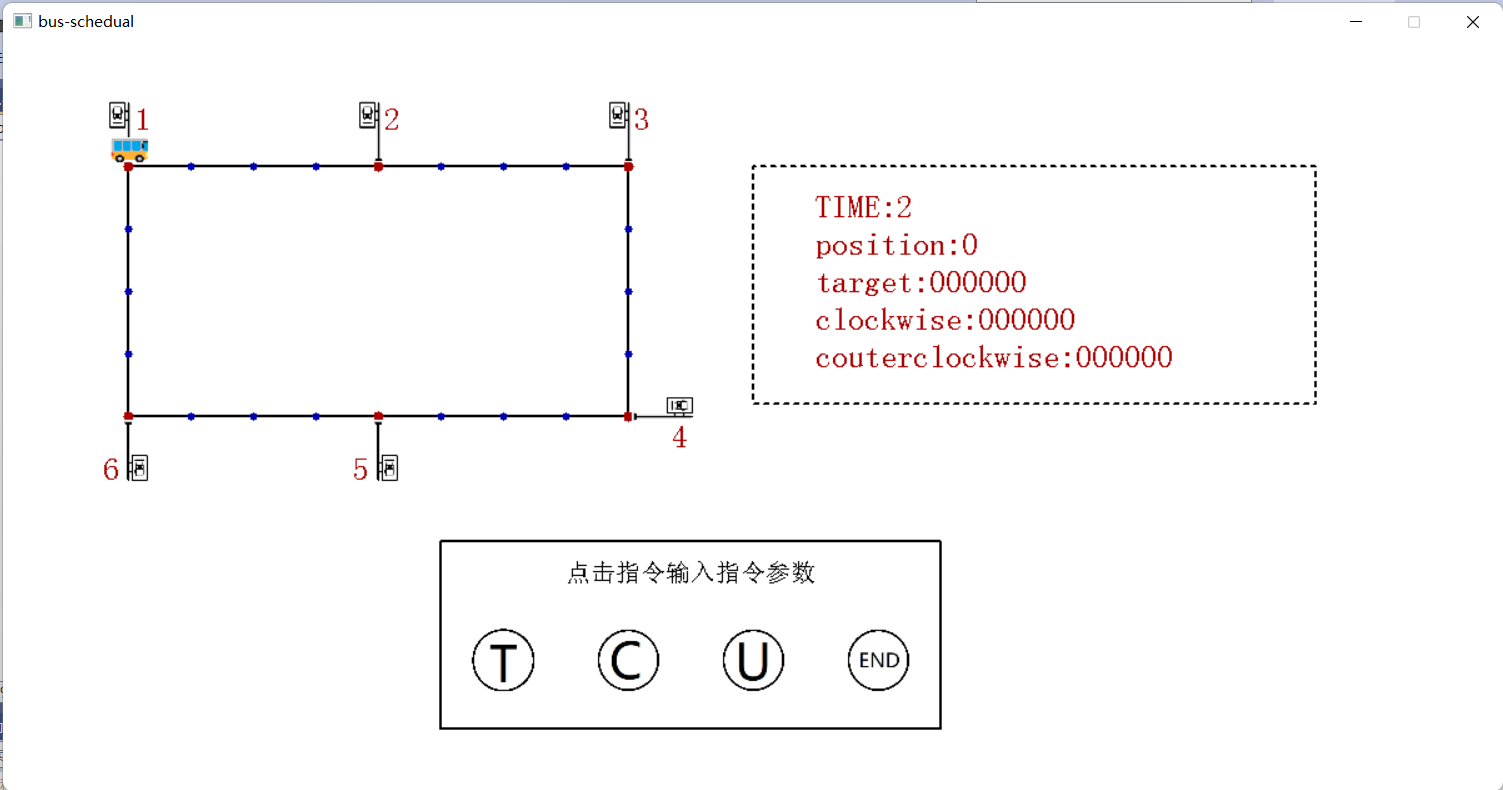
1. 公交车运行的动画显示：在动画区显示公路，车站，公交车的位置。



1. 公交车的状态显示：在输出区显示公交车收到的指令，当前的位置，当前的行驶方向，运行的时间。



1. **概要设计**
2. **用户界面设计**



本程序用户界面如图所示。

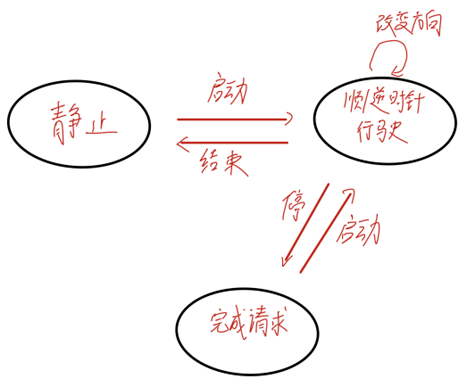
左侧是动画区。此道路中，汽车可以顺时针或逆时针行驶。此道路为封闭道路，可以循环行驶。道路中，每隔一定距离会有一个车站，车站有明显的标识，车站是公交车完成指令的地方。此区域会每个单位时间刷新一次，显示公交车的位置。

右侧是输出区，每个单位时间刷新一次，显示公交车的状态，包括运行时间，位置，已接受但为完成的指令。

下方是图形界面的输入区，有四个按钮，T，C，U，END，分别代表target，clockwise，counterclockwise，end四个指令，点击后输入数字（代表指令对应的车站）以发送指令给公交车。

1. **自动机模型（状态转换图）**

在本题目中，我们需要分析清楚汽车以及指令这两者的行为，在此我们使用状态图来进行描述。



1. **高层数据结构定义**

3.1 常量定义

// 指令类型常量设置

#define CMD\_CLOCK 0

#define CMD\_TARGET 1

#define CMD\_CLOCKWISE 2

#define CMD\_COUNTERCLOCKWISE 3

#define CMD\_END 4

// 策略类型常量设置

#define STRATEGY\_FCFS 1

#define STRATEGY\_SSTF 2

#define STRATEGY\_SCAN 3

// 最大车站数量

#define SIZE\_MAX\_STATION 25

3.2 数据结构定义

// 公交车结构体

typedef struct {

int station\_count; // 车站的数量

int station\_distance; // 车站间的距离

int total\_length; // 轨道的总长度

int strategy; // 调度策略

int time; // 当前时间

int direction; // 当前行驶方向

int position; // 当前位置

int clockwise[SIZE\_MAX\_STATION]; // 有顺时针请求的车站

int counter\_clock[SIZE\_MAX\_STATION]; // 有逆时针站请求的车站

int target[SIZE\_MAX\_STATION]; // 车内的请求

int cur\_command; // 正在调度的请求

int cur\_arg; // 正在调度的请求参数

int requests\_count;

ListNode\* requests\_list; // 已经发出的请求用链表保存

Queue\* requests\_queue; // 已经发出的请求用队列保存

int end;

} Bus;

//链表结构体

typedef struct Node {

int command;

int arg;

int time;

struct Node\* next;

} ListNode;

//队列结构体

typedef struct QNode {

int data;

struct QNode\* next;

} QueueNode;

typedef struct {

QueueNode\* front;

QueueNode\* rear;

int count;

} Queue;

1. **系统模块划分**

本系统程序部分划分为bus.cpp，bus-schedual.cpp，fcfs.cpp，gui.cpp，input.cpp，scan.cpp，sstf.cpp七个模块。各模块功能如下：

1.模块名称bus.cpp

              模块功能简要描述：进行与公交车相关操作的函数的定义。

2.模块名称bus\_schedual.cpp

模块功能简要描述：主函数，主要是运行各个子程序，实现公交车调度的动画化并响应用户的输入。

3.模块名称fcfs.cpp

模块功能简要描述：负责先来先服务策略的调度。

4.模块名称sstf.cpp

模块功能简要描述：负责最短寻找时间优先策略的调度。

5.模块名称scan.cpp

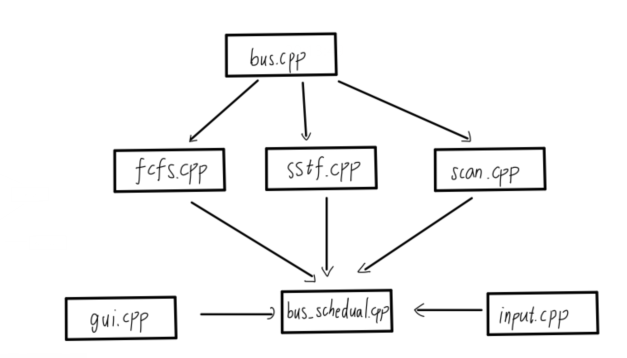
模块功能简要描述：负责最短寻找时间优先策略的调度。

6.模块名称gui.cpp

模块功能简要描述：负责绘制图形界面，接收用户点击和输入。

7.模块名称input.cpp

模块功能简要描述：读取配置文件和用户输入的指令。



1. **文件及函数组成**

|  |  |  |  |
| --- | --- | --- | --- |
| 源文件 | 源文件说明 | 函数名 | 功能 |
| bus.cpp | 与公交车操作相关的函数 | Bus\* NewBus(int total\_station, int distance, int strategy) | 创建并初始化公交车实例 |
| void ActionOnCommand(Bus\* bus, int command, int arg) | 执行请求并输出公交车状态 |
| void ActionOnCommandNoOutput(Bus\* bus, int command, int arg) | 执行请求但不输出公交车状态 |
| void ShowBusStatus(Bus\* bus) | 输出公交车状态 |
| void DeleteBus(Bus\* bus) | 释放公交车实例内存 |
| bus.h | 公交车结构体的定义和公交车操作相关函数的声明 | 无 | 无 |
| bus-schedual.cpp | 运行各个子程序，实现公交车调度的动画化并响应用户的输入。 | 无 | 无 |
| const.h | 定义常量 | 无 | 无 |
| fcfs.cpp | 负责先来先服务策略的调度 | void FCFSScheduleNewRequest(Bus\* bus) | 在FCFS策略中，对新请求进行调度 |
| void FCFSActionOnCommand(Bus\* bus, int command, int arg) | 在FCFS策略中，处理其他请求 |
| gui.cpp | 负责绘制图形界面，接收用户点击和输入。 | void DrawBusStatus(Bus\* bus) | 在图形界面输出公交车状态 |
| void DrawBoardWithBus(Bus\* bus) | 绘制道路、公交车、输入区和输出区 |
| unsigned int \_\_stdcall BackgroundClock(void \*p) | 计时并绘制图形 |
| void ReadCommandFromGUIAndAct(Bus\* bus) | 接收用户点击和输入并响应 |
| gui.h | 绘制图形界面相关函数的声明 | 无 | 无 |
| input.cpp | 读取配置文件和用户输入的指令 | void ReadInConfig(int\* station\_count, int\* station\_distance, int\* strategy) | 从dict.dic文件读取配置参数 |
| void ReadCommandAndAct(Bus\* bus) | 读入指令序列，并根据指令序列做出行为 |
| input.h | input.cpp中函数的声明 | 无 | 无 |
| list.cpp | 链表相关函数 | ListNode\* NewListNode(int command, int arg, int time, ListNode\* next) | 新建一个节点 |
| void FreeListNode(ListNode\* node) | 释放一个节点 |
| list.h | 链表结构体定义和链表相关函数声明 | 无 | 无 |
| queue.cpp | 队列相关函数 | Queue\* NewQueue() | 新建一个队列 |
| void Enqueue(Queue\* queue, int data) | 将值入队 |
| int Dequeue(Queue\* queue) | 出队 |
| int FrontQueue(Queue\* queue) | 取队头 |
| int SizeQueue(Queue\* queue) | 求队列的大小 |
| queue.h | 队列结构体定义和链表相关函数声明 | 无 | 无 |
| scan.cpp | 负责实现最短寻找时间策略 | void SCANDecideDirection(Bus\* bus) | 在SCAN策略中，决定公交车的方向 |
| void SCANActionOnCommand(Bus\* bus, int command, int arg) | 在SCAN策略中，执行请求 |
| sstf.cpp | 负责实现最短寻找时间策略 | void SSTFScheduleNewRequest(Bus\* bus) | 在SSTF策略中，对新请求进行调度 |
| void SSTFActionOnCommand(Bus\* bus, int command, int arg) | 在SSTf策略中，执行其他请求 |

1. **函数说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | Bus\* NewBus(int total\_station, int distance, int strategy) | 创建并初始化公交车实例 | total\_station代表总的车站数，distance代表车站间的距离，strategy代表策略 | Bus类型的结构体 |
| 2 | void ActionOnCommand(Bus\* bus, int command, int arg) | 执行请求并输出公交车状态 | bus代表公交车，command代表请求，arg代表请求的车站编号 | void |
| 3 | void ActionOnCommandNoOutput(Bus\* bus, int command, int arg) | 执行请求但不输出公交车状态 | bus代表公交车，command代表请求，arg代表请求的车站编号 | void |
| 4 | void ShowBusStatus(Bus\* bus) | 输出公交车状态 | bus代表公交车 | void |
| 5 | void DeleteBus(Bus\* bus) | 释放公交车实例内存 | bus代表公交车 | void |
| 6 | void FCFSScheduleNewRequest(Bus\* bus) | 在FCFS策略中，对新请求进行调度 | bus代表公交车 | void |
| 7 | void FCFSActionOnCommand(Bus\* bus, int command, int arg) | 在FCFS策略中，处理请求 | bus代表公交车，command代表请求，arg代表请求的车站编号 | void |
| 8 | void DrawBusStatus(Bus\* bus) | 在图形界面输出公交车状态 | bus代表公交车 | void |
| 9 | void DrawBoardWithBus(Bus\* bus) | 绘制道路、公交车、输入区和输出区 | bus代表公交车 | void |
| 10 | unsigned int \_\_stdcall BackgroundClock(void \*p) | 计时并绘制图形 | p为一空类型指针 | unsigned int |
| 11 | void ReadCommandFromGUIAndAct(Bus\* bus) | 接收用户点击和输入并响应 | bus代表公交车 | void |
| 12 | void ReadInConfig(int\* station\_count, int\* station\_distance, int\* strategy) | 从dict.dic文件读取配置参数 | station\_count代表车站数，station\_distance代表车站间距离，strategy代表策略 | void |
| 13 | void ReadCommandAndAct(Bus\* bus) | 读入指令序列，并根据指令序列做出行为 | bus代表公交车 | void |
| 14 | ListNode\* NewListNode(int command, int arg, int time, ListNode\* next) | 新建一个节点 | command代表请求，arg代表请求的参数，time代表时间，next代表下一节点 | ListNode类型的结构体 |
| 15 | void FreeListNode(ListNode\* node) | 释放一个节点 | node代表要释放的节点 | void |
| 16 | Queue\* NewQueue() | 新建一个队列 | 无 | Queue类型的结构体 |
| 17 | void Enqueue(Queue\* queue, int data) | 将值入队 | queue代表队列，data代表值 | void |
| 18 | int Dequeue(Queue\* queue) | 出队 | queue代表队列 | int |
| 19 | int FrontQueue(Queue\* queue) | 取队头 | queue代表队列 | int |
| 20 | int SizeQueue(Queue\* queue) | 求队列的大小 | queue代表队列 | int |
| 21 | void SCANDecideDirection(Bus\* bus) | SCAN策略中，决定公交车的方向 | bus代表公交车 | void |
| 22 | void SCANActionOnCommand(Bus\* bus, int command, int arg) | SSTF策略中处理请求 | bus代表公交车，command代表请求，arg代表请求的车站编号 | void |
| 23 | void SSTFScheduleNewRequest(Bus\* bus) | SSTF策略中对新请求进行调度 | bus代表公交车 | void |
| 24 | void SSTFActionOnCommand(Bus\* bus, int command, int arg) | SSTF策略中处理请求 | bus代表公交车，command代表请求，arg代表请求的车站编号 | void |

1. **高层算法设计**

下面给出该工程中几个高层算法的伪代码：

void SCANActionOnCommand(Bus \*bus, int command, int arg) {

    把命令中的站点序号从 1-10 变成 0-9

    if  command 为 CMD\_CLOCK {

        时间加一

        if 公交车到达站点位置{

             求站序号

             判断是否需要在该站停车

             把所有该站点的已经发出的请求都处理完

}

            if 需要停车 {

                完成清空此处的所有请求

                return；

}

        判断是否需要改变公交车的方向

        if 需要改变方向{

改变方向

}

 if 不需要改变方向{

继续

}

    if  command 为 CMD\_TARGET {

        if 公交车的target[arg]不为0

return;

            公交车的target[arg] <-1;

}

    else if command 为 CMD\_CLOCKWISE{

        if 公交车的clockwise[arg]不为0

return;

              公交车的clockwise[arg] <- 1;

}

    else if command 为 CMD\_COUNTERCLOCKWISE{

        if 公交车的counter\_clock[arg]

return;

              公交车的counter\_clock[arg] =<-  1;

}

    更新指令

}

void SSTFActionOnCommand(Bus \*bus, int command, int arg) {

把命令中的站点序号从 1-10 变成 0-9

if  command 为 CMD\_CLOCK

      处理此时的clock请求

if command 为CMD\_TARGET{

          if 公交车的target[arg]不为0

return;

            公交车的target[arg] <-1;

    }

else if command 为 CMD\_CLOCKWISE{

        if 公交车的clockwise[arg]不为0

return;

              公交车的clockwise[arg] <- 1;

    }

else if command 为 CMD\_COUNTERCLOCKWISE{

        if 公交车的counter\_clock[arg]

return;

              公交车的counter\_clock[arg] =<-  1;

    }

   更新指令

}

void FCFSScheduleNewRequest(Bus \*bus) {

    if（请求队列为空）{

        公交车的方向和指令赋值为0（空）

        返回

    }

    取出队首

    把队首赋值给公交车的当前指令和对应位置

    计算目标位置 target\_pos

    计算和目标的顺时针距离 clockwise\_distance

    if（clockwise\_distance 为负） {

              给clockwise\_distance加上总路程，变为正

       }

    if （clockwise\_distance小于等于一半总路程）{

              公交车的方向为顺时针

       }