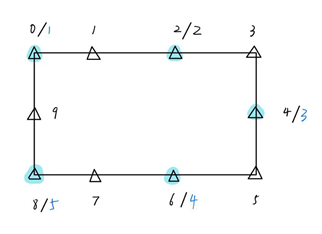
**概要设计**

1. **用户界面设计**

 （以站点总数为10，每站之间的距离为2为例）

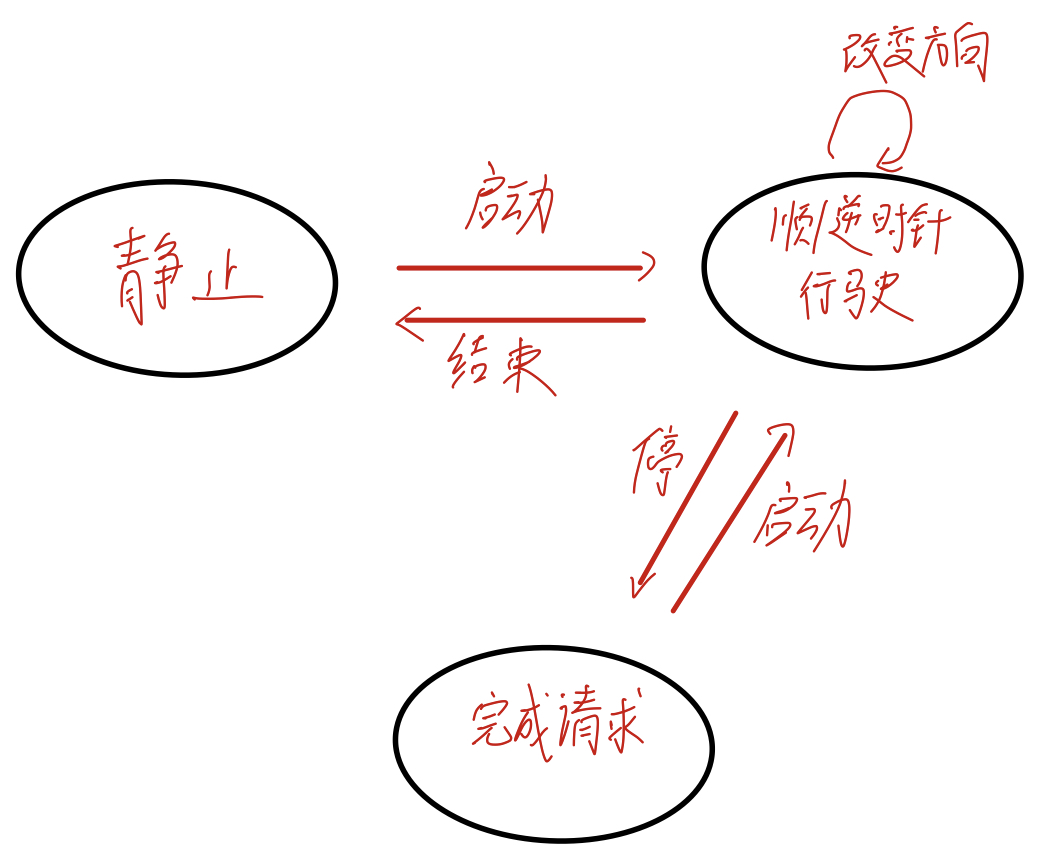
本程序用户界面设计如上图所示（普通三角为坐标，蓝色三角同时为车站位置）。各站之间距离可配置，站点个数可配置，调度策略可配置。这三个参数保存在配置文件中，程序要通过读配置文件获取。在此环形轨道中，车辆可以朝两个方向行驶，顺时针或逆时针。

当用户控制公交车运行时，指令由键盘输入。指令共5种。分别为end、clock、counterclockwise、clockwise 和target。解释如下：

1. 输入‘clock’表示时间过了一秒钟
2. 输入‘end’表示时钟停止，程序退出
3. 输入‘counterclockwise x（x代表常数，下同）’表示逆时针方向乘车请求，后边的整数代表请求发生的站点号
4. 输入‘clockwise x（x代表常数，下同）’表示顺时针方向乘车请求，后边的整数代表请求发生的站点号
5. 输入‘target x’表示车厢内下车请求，后边的整数代表要去的站点号

**2.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。



**3.高层数据结构设计**

**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

// 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全局变量定义**

无

**3.3数据结构定义**

//链表结构体

typedef struct QNode {

int data;

struct QNode \*next;

} QueueNode;

//队列结构体

typedef struct {

QueueNode \*front;

QueueNode \*rear;

int count;

} Queue;

//指令链表结构体

typedef struct Node {

int command;

int arg;

int time;

struct Node \*next;

} ListNode;

// 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

} Bus;

**4.系统模块划分**

**4.1软件结构图**

本系统程序部分划分为 head.h ，main.c，FCFS.c，SSTF.c，SCAN.c，五个模块。各模块功能如下：

1. 模块名称head.h

模块功能简要描述：进行常量、数据结构的定义，部分函数的说明。

1. 模块名称main.c

模块功能简要描述：主函数，主要是运行各个子程序，控制方案的调度。

1. 模块名称FCFS.c

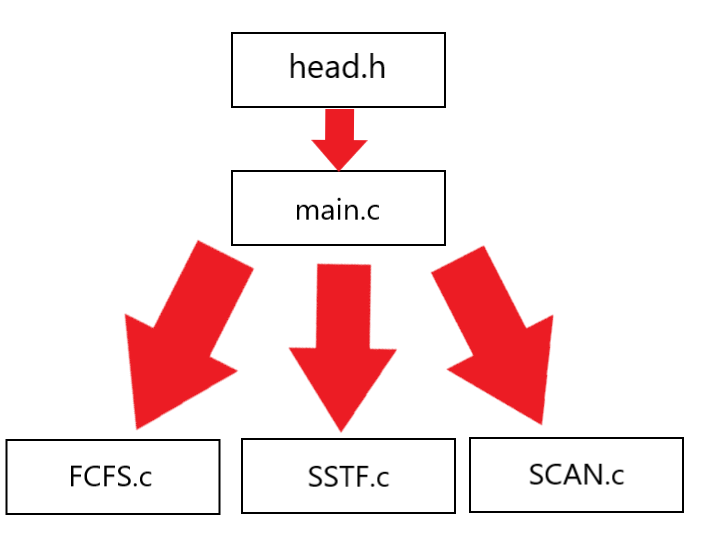
模块功能简要描述：负责先来先服务策略的进行。

1. 模块名称SSTF.c

模块功能简要描述：负责最短寻找时间优先策略的进行。

1. 模块名称SCAN.c

模块功能简要描述：负责顺便服务策略的进行。



**4.2文件及函数组成**

|  |  |  |  |
| --- | --- | --- | --- |
| 源文件 | 源文件说明 | 函数名 | 功能 |
| main.c | 主函数，主要是运行各个子程序，控制方案的调度。 | void ActionOnCommand(Bus \*bus, int command, int arg) | 针对不同的策略，选择不同的函数处理请求 |
| 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 |
| head.h | 进行常量、数据结构的定义，部分函数的说明。 | Queue\* NewQueue() | 新建一个队列 |
| void Enqueue(Queue \*queue, int value) | 将值入队 |
| int Dequeue(Queue \*queue) | 出队 |
| int FrontQueue(Queue \*queue) | 取队头 |
| 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 |
| Bus\* NewBus(int total\_station, int distance, int strategy) | 创建并初始化公交车实例 |
| void ShowBusStatus(Bus \*bus) | 输出公交车状态 |
| void DeleteBus(Bus \*bus) | 释放公交车实例内存 |
| FCFS.c | 负责实现先来先服务策略 | void FCFSScheduleNewRequest(Bus \*bus) | FCFS策略中对新请求进行调度 |
| void FCFSActionOnClock(Bus \*bus) | FCFS策略中实现对clock的处理 |
| void FCFSActionOnCommand(Bus \*bus, int command, int arg) | FCFS策略中实现对其他请求的处理 |
| SSTF.c | 负责实现最短寻找时间策略 | void SSTFScheduleNewRequest(Bus \*bus) | SSTF策略中对新请求进行调度 |
| int SSTFAchieveStation(Bus \*bus) | SSTF策略中判断公交车是否到站 |
| void SSTFActionOnClock(Bus \*bus) | SSTF策略中处理clock请求 |
| void SSTFActionOnCommand(Bus \*bus, int command, int arg) | SSTF策略中处理其他请求 |
| SCAN.c | 负责实现顺便服务策略 | void SCANDecideDirection(Bus \*bus) | SCAN策略中，决定公交车的方向 |
| void SCANActionOnCommand(Bus \*bus, int command, int arg) | SCAN策略中，执行请求 |

**4.3函数说明**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 函数原型 | 功能 | 参数 | 返回值 |
| 1 | Queue\* NewQueue() | 新建一个队列 | 无 | Queue结构体 |
| 2 | void Enqueue(Queue \*queue, int value) | 将值入队 | queue是队列，value是入队的值 | void |
| 3 | int Dequeue(Queue \*queue) | 出队 | queue是进行出队操作的队列 | 出队的值 |
| 4 | int FrontQueue(Queue \*queue) | 取队头 | queue是进行取队头操作的队列 | 队头的值 |
| 5 | 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结构体 |
| 6 | void FreeListNode(ListNode \*node) | 释放一个节点 | node代表要释放的节点 | void |
| 7 | 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 |
| 8 | void ShowBusStatus(Bus \*bus) | 输出公交车状态 | bus代表公交车 | void |
| 9 | void DeleteBus(Bus \*bus) | 释放公交车实例内存 | bus代表公交车 | void |
| 10 | void FCFSScheduleNewRequest(Bus \*bus) | FCFS策略中对新请求进行调度 | bus代表公交车 | void |
| 11 | void FCFSActionOnClock(Bus \*bus) | FCFS策略中处理clock请求 | bus代表公交车 | void |
| 12 | void FCFSActionOnCommand(Bus \*bus, int command, int arg) | FCFS策略中处理其他请求 | bus代表公交车, command代表指令， arg代表请求的参数 | void |
| 13 | void SSTFScheduleNewRequest(Bus \*bus) | SSTF策略中对新请求进行调度 | bus代表公交车 | void |
| 14 | int SSTFAchieveStation(Bus \*bus) | SSTF策略中判断公交车是否到站 | bus代表公交车 | 1代表到站，0代表未到站 |
| 15 | void SSTFActionOnClock(Bus \*bus) | SSTF策略中处理clock请求 | bus代表公交车 | void |
| 16 | void SSTFActionOnCommand(Bus \*bus, int command, int arg) | SSTF策略中处理其他请求 | bus代表公交车, command代表指令， arg代表请求的参数 | void |
| 17 | void SCANDecideDirection(Bus \*bus) | SCAN策略中，决定公交车的方向 | bus代表公交车 | void |
| 18 | void SCANActionOnCommand(Bus \*bus, int command, int arg) | SCAN策略中，执行请求 | bus代表公交车command代表请求，arg代表请求的参数 | void |
| 19 | void ActionOnCommand(Bus \*bus, int command, int arg) | 针对不同的策略，选择不同的函数处理请求 | bus代表公交车command代表请求，arg代表请求的参数 | void |
| 20 | 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 |
| 21 | 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 |

**5.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

}

else{

公交车的方向方向为逆时针

}

}