本科生毕业设计[论文]

基于离散型街道垃圾处理系统的模拟仿真实现与算法设计

院　　系　计算机学院

专业班级　计算机科学与技术二学位

姓　　名　李文扬

学　　号　U201416091

指导教师　姚 杰

2017 年 8 月 29 日

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密□，在　　年解密后适用本授权书

2、不保密□　。

（请在以上相应方框内打“√”）

作者签名：　　年　月　日

导师签名：　　年　月　日

摘　要

目前，计算机科学技术与大数据技术的迅速发展带动了各个产业领域的发展与变革。伴随着智慧城市概念的提出，像物联网技术，遥感技术，无人机技术，信息决策支持系统等领域得到快速发展。同样，随着城市规模的扩大，各种有关城市建设与维护的问题也越来越重要。

其中一个典型的问题就是城市街道垃圾处理的问题。城市规模的扩大以及人口密度的不断增加，导致城市街道垃圾的处理越来越棘手。而如何能够基于大数据信息动态，智能地调度环卫工，实现在较少的资源利用下，实现城市垃圾及时清理的目的正是文章所要研究的方向。

通过对传统垃圾处理流程的调研，利用离散仿真模拟，并行控制的消息传递以及面向对象设计等相关技术。在明确系统的设计需求与功能需求的前提下，搭建一个能够模拟在一定区域内城市街道垃圾的处理系统框架。该系统能够对一定区域内的环卫工进行实时的调度，实现对垃圾处理的实时管理，并能够产生相应的分析数据以供分析。

同时，文章也讨论分析了基于本框架下，应用不同的调度算法时（传统的顺序调度算法 随机调度算法 基于区块划分的调度算法 基于距离的调度算法）对整个系统性能的影响。

最后，通过探讨本框架的不足之处，对本模型可以改进的地方进行了分析。并做了本模型在实际应用中的可行性分析，探讨了该系统下一步的发展方向。

关键词：垃圾处理，离散仿真系统,并发控制，调度优化，面向对象设计思想。

Abstract

Nowadays, the rapid development of computer science & technology and big data technology has led to the development and transformation of various industrial fields. With the introduction of the concept of intelligent city, such as Internet of things technology, remote sensing technology, unmanned aerial vehicle technology, information decision support system and other fields have been rapid development. One of the typical problems is the problem of urban street rubbish disposal. The expansion of urban scale and the increasing population density have led to increasingly difficult handling of urban street waste. And how can be based on big data information dynamic and intelligent scheduling sanitation workers, to achieve the use of less resources, to achieve the purpose of timely disposal of municipal waste is the direction of this paper to be studied.

In this paper, trough the traditional rubbish disposal process researching, with the use of discrete simulation, parallel control of the message delivery and object-oriented design and other related technologies. In a clear system design requirements and functional requirements under the premise of building a can be simulated in a certain area of urban street garbage processing system framework. The system can be a certain area of sanitation workers in real-time scheduling, real-time management of garbage disposal, and can produce the appropriate analysis of data for analysis.

At the same time, this paper also discusses the analysis of the framework based on the application of different scheduling algorithm (1. The traditional sequential scheduling algorithm 2. Random scheduling algorithm 3. Block based on the scheduling algorithm 4. Distance-based scheduling algorithm) At last, this paper discusses the shortcomings of this framework and analyzes the areas where the model can be improved. And the feasibility analysis of this model in practical application is made, and the development direction of the system is discussed.

Key Words： Rubbish Disposal, Discrete Simulation System, Concurrent Control, Scheduling Optimization, Object - Oriented Design

目 录

1 绪论

1.1 课题研究背景及意义

目前，计算机科学技术的迅猛发展带动了各产业领域的信息化变革。伴随着大数据技术，人工智能技术以及物联网等相关概念的提出，如何实现智能化的生产运作与管理调度成为了当下的一个热门研究领域。

随着城市规模的不断扩大，人口密度的不断增加以及城市之间互动的频率加快，一方面反映了经济的高速发展，但另一方面也给居民的生活带来了诸多不便。过去单一的人工管理方式已经远远无法满足当前城市管理的需求。这就要求能够使用信息管理技术来实现城市的智能管理。

在该领域中，如何有效地进行城市管理调度一直以来是一个热点课题。“智慧城市”是利用计算机技术以及信息管理技术等手段，对城市的环保，民生，城市服务，公共安全，工商业活动在内的各种需求做出智能响应，使城市成为一个有机的实体。这一概念最早是在 2008 年 11 月，在纽约召开的外国关系理事会上，由 IBM 提出的“智慧的地球” 这一理念演化发展而来。而在 2010 年，IBM 正式提出了“智慧的城市”这一愿景，标志着这一概念的正式形成。

表 1-1“智慧城市”时间线

时间	发展历程
2002-2005 年	欧洲实施了“电子欧洲”行动计划
2006 年	新加坡启动“智慧国 2015”计划，旨在将新加坡打造为信息化的一流国家
2009 年	迪比克市与 IBM 合作，建立美国第一个智慧城市
2009 年	日本推出“I-Japan 智慧日本战略 2015”

目前，我国政府也开始逐步实施智慧城市打造计划，智慧上海，智慧双流等计划已经开始实施并取得了一系列的成果。

在智慧城市领域中，一个核心的问题就是城市垃圾的管理，这个问题涉及到了居民的基本生活。现在，各种节日活动的举办，以及旅游热潮的兴起，导致大型城市的人口流动十分频繁。这种情况导致了城市中街道垃圾处理不及时的问题。过去，为了解决这一问题，政府一般会投入大量的人力物力，但这种方法需要的成本过大，经常会造成资源的浪费，导致处理效率低下以及响应能力较差的问题。

为了能够更好地解决这一问题，需要构建一个能够进行自动实时进行人员调度分配的垃圾处理系统。通过使用较少的环卫工，达到实时垃圾处理的目的，而这也是文章的主要研究课题。

1.2 国内外研究现状

目前业界对于“智慧城市”的理解从概念上讲有以下四个主要特征：

——全面透彻的感知：是指通过传感器技术，对城市管理的各个方面进行实时监测，从而能在第一时间获得有效信息，为实现城市管理的实时响应做好基础。

——宽带泛在的互联：通过网络技术进行数据的共享，从而实现城市各模块实体之间的实时互联，使城市以及城市与城市之间成为一个有机的实体。

——智能融合的应用：伴随着城市规模的不断扩大，基于云计算与大数据处理技术，实现海量数据的处理。并提高系统的耦合度，实现从“云”到“端”的实时联系。

——以人为本的可持续创新：提高系统的可维护性以及可扩展性，设计用户友好的管理系统。关注市民基本需求，实现可持续发展。

而关于智慧城市的实践可以追溯到 2006 年欧盟发起的欧洲 Living Lab 组织。它采用新的工具和方法、先进的信息和通讯技术来调动方方面面的“集体的智慧和创造力”，为解决社会问题提供机会。该组织还发起了欧洲智慧城市

网络。Living Lab 完全是以用户为中心，借助开放创新空间的打造帮助居民利用信息技术和移动应用服务提升生活质量，使人的需求在其间得到最大的尊重和满足。

之后在 2009 年，日本和美国相继启动了自己的“智慧城市”计划。可以说，到目前为止，国内外在“智慧城市”领域有着十分高的研究热度。

而城市垃圾处理作为“智慧城市”中的重要研究方向，一直以来有着很高的关注度。根据相关文献资料显示，目前关于城市垃圾处理的研究大致可以分为三个方向：

第一个方向是面向终端的智能设备研究，该方向的研究重点在于如何设计智能终端设备，如手持式扫描器：一种可以与调度系统实时进行交互的设备。智能垃圾桶：可以智能的检测自身负载状况并及时做出汇报。

而该领域的第二个方向的研究重点在于如何设计垃圾的实时调度系统，而垃圾的调度系统的设计又可以分为居民生活垃圾管理调度，工业垃圾管理调度，有害垃圾管理调度系统等。

第三个方向是关于如何有效地将垃圾处理模块嵌入到整个“智慧城市”的生态圈中，以实现个模块之间信息的有效互通以及资源共享。

1.3 本文主要内容

本文的主要研究课题是基于离散型街道垃圾处理系统的模拟仿真实现与算法设计。文章旨在设计一套垃圾系统调度框架，它涉及到对垃圾产生时的监测，垃圾处理的调度分配，垃圾处理过程的实时监控以及垃圾处理信息的输出分析等一整套流程的仿真模拟。并设计了四种调度算法来实现垃圾与环卫工的调度匹配。

全文内容共分为以下几大章节：

第一章为绪论，介绍了本文所要研究的课题内容以及目前城市垃圾处理系统发展的现状。

第二章为系统设计相关技术，介绍了开发本模拟仿真系统所需要的相关技

术与开发工具。

第三章为需求分析与概要设计，主要对整个垃圾处理流程进行分析并对各实体之间的关系抽象化，完成本系统的整体的概要逻辑设计。

第四章为系统详细设计与实现，基于上一章分析，实现整个系统的构建，并设计相关算法实现环卫工的具体调度。

第五章为系统测试结果分析，主要是测试本模拟仿真系统的运行效果以及对不同调度算法的运行结果进行对比。

第六章为展望与总结，主要对整个开发过程进行一个回顾，对系统中的一些待改进的地方进行了分析，并对进一步的开发工作进行了展望。

1.4 本章小结

本章主要介绍了本文的研究背景以及主要内容，并介绍了本文各部分之间的内容关系。整个模拟仿真系统的开发细节以及测试分析将在后面的各章节中进行详细介绍。

2 系统设计相关技术

2.1 离散系统仿真技术

计算机仿真技术（Computer simulation technique）是指在不对实体系统进行操作的情况下，应用计算机来对该实体系统进行模拟动态实验的技术。仿真技术最早发源于军事领域，由于它具有成本低，真实性强，安全性高，不受极端情况影响等优点，仿真技术被迅速应用于商业以及民用领域。

计算机仿真技术有主要分为两大类：连续事件系统（Continuous system）的仿真方法和离散事件系统（Discrete system）的仿真方法。两者的主要差异在于系统中事件的特性在时间上表现为连续型事件还是离散型事件。

离散仿真系统的主要特点是事件的发生在时间上是离散的，一般是基于某种概率分布。系统会根据各事件之间的关系，模拟整个系统的运作过程。离散事件系统仿真广泛用于交通管理、生产调度、资源利用、计算机网络系统的分析和设计方面。一般来说，离散型系统的模拟一般有以下几个步骤：

1.画出系统的工作流程图（Flow chart）;

2.确定到达模型（Arrive model）、服务模型（Service Model）和排队模型（Queuing Model），它们构成离散事件系统的仿真模型；

3.编制描述系统活动的运行程序；

4.在计算机上执行这个程序；

5.统计分析模拟过程中产生的各数据。

基于现实生活中垃圾产生与处理的时间特性是离散的，所以本文所用到的仿真原理是离散事件系统的仿真方法。

2.2 并发控制机制

并发性（Concurrency）是指在同一时间间隔内有两个或多个事件发生的情况，这种情况在现实世界中十分常见。目前随着计算机技术的发展，已经可以实现各进程之间的并行运行。

而在计算机科学领域，并发控制机制（Concurrency control mechanism）是确保及时纠正由并发操作导致的错误的一种机制。它可以防止由于不同进程事务之间的相互影响而导致数据处理错误或程序运行混乱的情况发生。目前来说，并行控制技术的应用面很广，像数据库管理系统，军事管理系统，计算机操作系统等领域都有一套完整成熟的并发控制机制。

目前来说，较成熟的并发控制技术有

1. 锁机制（Locking），通常用于对互斥型变量的访问与修改，在数据库管理系统中应用较广，目前应用较为广泛的是三级封锁协议。

表 2-1 三级封锁协议示意图

	X 锁	S 锁		无丢失修改	无脏读	可更新
	事务结束释放	事务结束释放	读完释放			
一级封锁协议	√			√		
二级封锁协议	√		√	√	√	
三级封锁协议	√	√		√	√	√

2.信号灯机制（Semaphore），通常用于限制可以访问某些资源（物理或逻辑的）的线程数目，在操作系统的进程调度方面应用较为广泛。

2.3 算法分析原理

算法分析（Algorithm analyze）是对一个算法需要多少计算时间和存储空间作定量的分析。考虑到算法的时间复杂度（Time complexity）与空间复杂度（Space complexity）是可以相互转化的，所以一般来说，我们更关注的是一个算法的时间复杂度。而对于时间复杂度，我们往往关注一个算法随着输入规模的增长，其增长率的变化。

刻画算法时间复杂度的数学工具有很多，本文使用渐进公式O(n)来衡量，它表示一个算法在最差输入的情况下所消耗的时间资源。这种方法可以有效，清晰地刻画整个算法的时间复杂度。

2.4 面向对象设计技术

面向对象设计技术（Object oriented）是一种软件开发技术。它有别于传统的面向过程的开发技术（Procedure Oriented），它也是为了解决软件危机所发展出来的一种设计思想。

简单来说，通过将程序封装为一系列的类，使程序的开发与设计更贴近于人类的思维方式，更便于后期的维护与调试。在实时交互系统，信息管理系统，图形处理系统等领域，面向对象的设计技术也体现出了它的绝对优势。面向对象设计技术的典型特征一般有以下几点：

1.封装性(Encapsulation)：通过对对象进行封装，将各对象的关键数据保护起来。用户通过程序提供的接口来操作对象，这大大的提高了程序的安全性于易维护性。

2.继承性（Inheritance）：继承性是子类自动共享父类数据和方法的机制。通过继承机制，程序可以很容易地进行扩展与维护。

3.多态性（Polymorphism）：同一个方法在不同的类中有不同的实现方式，这就是多态性，一般是通过重载技术来实现多态。

目前，大部分的主流程序设计语言如C++ ,Java ,C# ,Python等，都支持面向对象设计机制。

2.5 本章小结

本章主要介绍了在搭建垃圾处理系统过程以及性能分析中所涉及到的一些关键技术与思想。像离散仿真技术，面向对象设计思想以及并发控制技术在本系统的设计过程中都有所运用，而算法的时间复杂度分析则在性能比较部分作为主要的评判标准工具。

3 需求分析与概要设计

3.1 系统需求分析

本系统主要的设计目标是能够实现现实生活中城市街道垃圾处理过程的模拟。但考虑到现实世界中各种不确定因素的影响，该系统不能实现对现实生活中垃圾处理的完全模拟。但通过对整个垃圾处理流程的抽象，该系统的重点在于实现对垃圾处理系统的基本流程模拟与功能实现。通过分析，该系统应具有如下功能：

（1）垃圾产生功能（到达模型模块）：本系统要能够按照一定的时间间隔，在给定区域内随机地产生垃圾，在这里垃圾的产生时间与产生地点都是随机的。在产生完垃圾后，系统能够记录下每一个垃圾的相关信息，便于后续进程处理。

（2）环卫工管理功能：本系统要能够对区域内的环卫工进行实时管理，以便于在后续调度分配阶段的时候，能够及时的获得环卫工的位置以及工作状态等信息。

（3）垃圾分配调度功能（服务模块）：当垃圾产生后，系统需要根据垃圾的位置信息以及环卫工的相关信息，进行相应的匹配调度。针对不同的调度算法，系统的垃圾分配调度机制要有不同的实现方式。

（4）异常处理功能：考虑到在系统运行过程中可能会出现一个垃圾被分配给多个环卫工，或者出现死锁的异常情况。所以，系统要有能够预防这种情况发生的机制。

（5）数据统计功能：因为本系统不仅要能够对整个处理流程进行仿真模拟，还要能够产生相应的统计数据来提供给用户，以便于用户来进行评估比对。一般来说，所需要用到的数据有垃圾的存留时间，环卫工的工作时间，工作强度等数据。

3.2 系统概要设计

因为在现实世界中，城市街道垃圾的调度处理是由很多因素决定的，其中也有很多不确定因素。要全面地考虑到所有的情况是不现实的，所以本系统在构建前要基于如下几个假设：

（1） 垃圾的产生是随机并且稳定的，且不考虑突发情况（如大型户外活动，极端天气等）对垃圾产生的影响。

（2） 垃圾的特征信息是同质的，即在本系统中所有的垃圾都是不可移动，不可分解，并且有相同的处理时间。

（3） 本系统中环卫工的数量是有限的，并且每个环卫工的特征信息也是相同的，有相同的移动速度，相同的处理速度。在系统运行期间内，假设环卫工一直处在上班状态，并且只要没有收到指令，就不会自己移动。

（4） 系统的调度信息是即时发送的，换句话说，当调度信息产生后，环卫工可以立即收到调度信息，并据此做出下一步行为。

（5） 在整个模拟区域内，是没有阻挡物（河流，建筑，池塘等）的，环卫工可以到达区域内任何指定地点。

（6） 整个区域内的地理位置信息均为整型数，这正好与显示屏幕的像素点对应，这样可以简化输出信息的处理。

基于以上的六条假设，从整体上来说，整个模拟系统的运行流程如图3-1 所示：

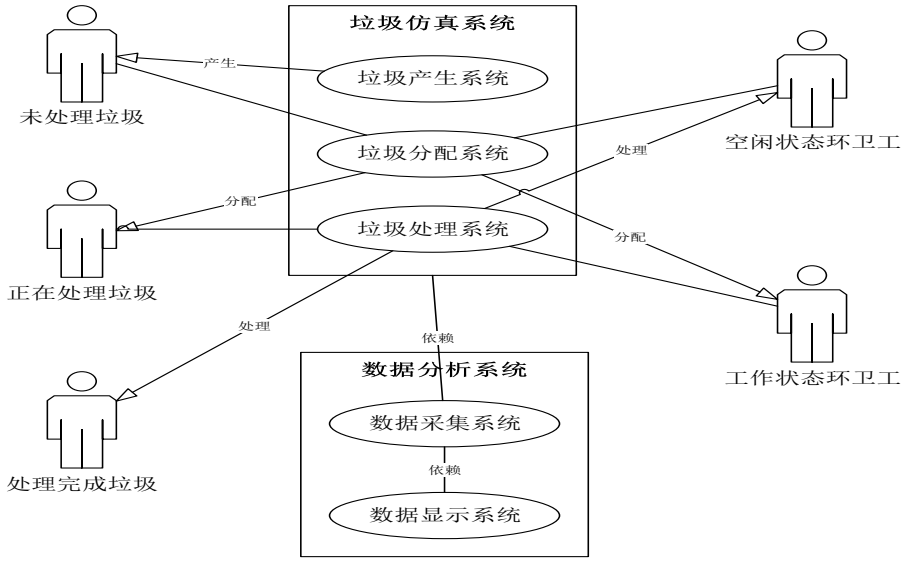

图 3-1 离散型垃圾调度仿真系统用例图

为了实现上述的流程，整个模拟系统的基本框架采用模块式的开发方式，如图 3-2 所示：

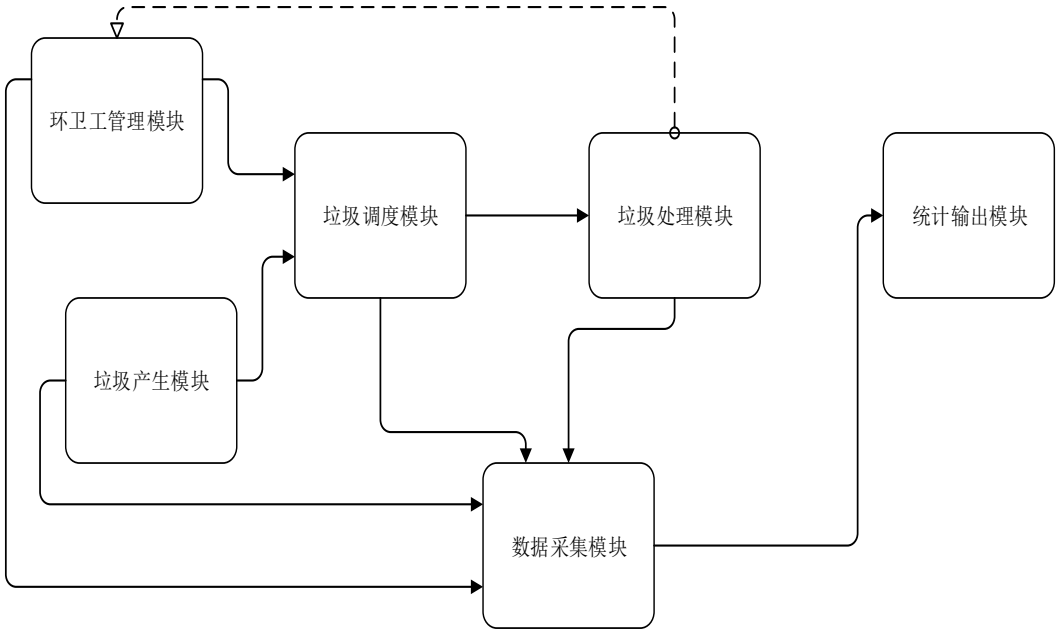

图 3-2 垃圾调度仿真系统模块示意图

为了能够实现对垃圾调度过程的实时模拟，本系统采用模块化的设计方式。整个框架被分为了六大模块，而各模块之间是可以并行运行的，它们

之间通过共享变量的方式来实现实时交互的目的。

3.3 各模块主要功能与流程图

3.3.1 垃圾产生模块

该模块主要负责整个模拟系统中垃圾的产生，并将所产生垃圾的基本信息（地理信息，时间信息）提交后续模块。

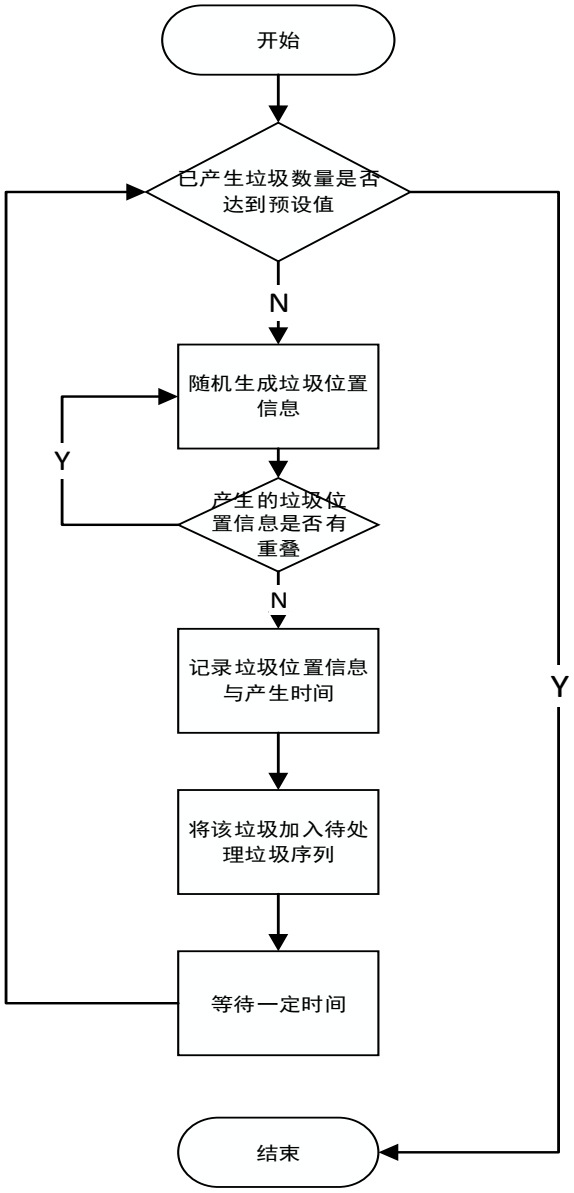

图 3-3 垃圾产生模块流程图

3.3.2 环卫工管理模块

该模块主要负责对模拟系统中环卫工的信息初始化以及后续信息的管理。

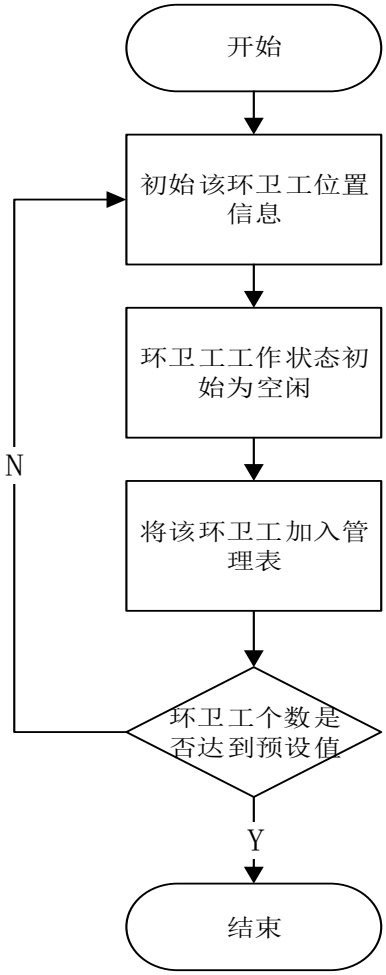

图 3-4 环卫工管理模块流程图

3.3.3 垃圾调度模块

该模块是整个模拟系统的核心模块，该模块利用前面垃圾产生模块以及环卫工管理模块所提供的信息，并利用给定的调度算法产生相应的调度分配决定。环卫工管理模块接收到相应调度信息后，对相应环卫工进行分配设定，以达到实现环卫工的实时调度的目的。

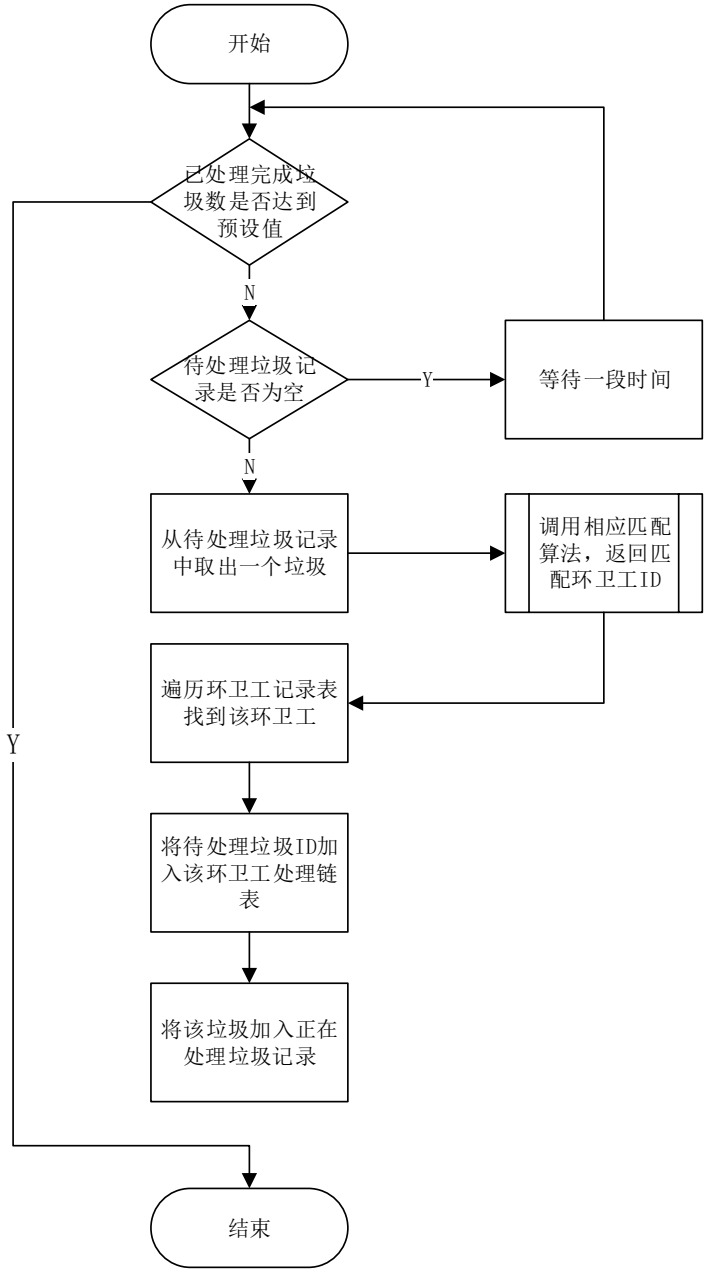

图 3-5 垃圾调度模块流程图

3.3.4 垃圾处理模块

在完成分配以及调度处理后，垃圾处理模块负责管理每个正在被处理的垃圾，并记录每个垃圾被处理的时间，一旦达到垃圾处理时间的预设值，就表明垃圾处理完成，通知环卫工管理模块进行相应的处理，完成整个垃圾处理的流程。

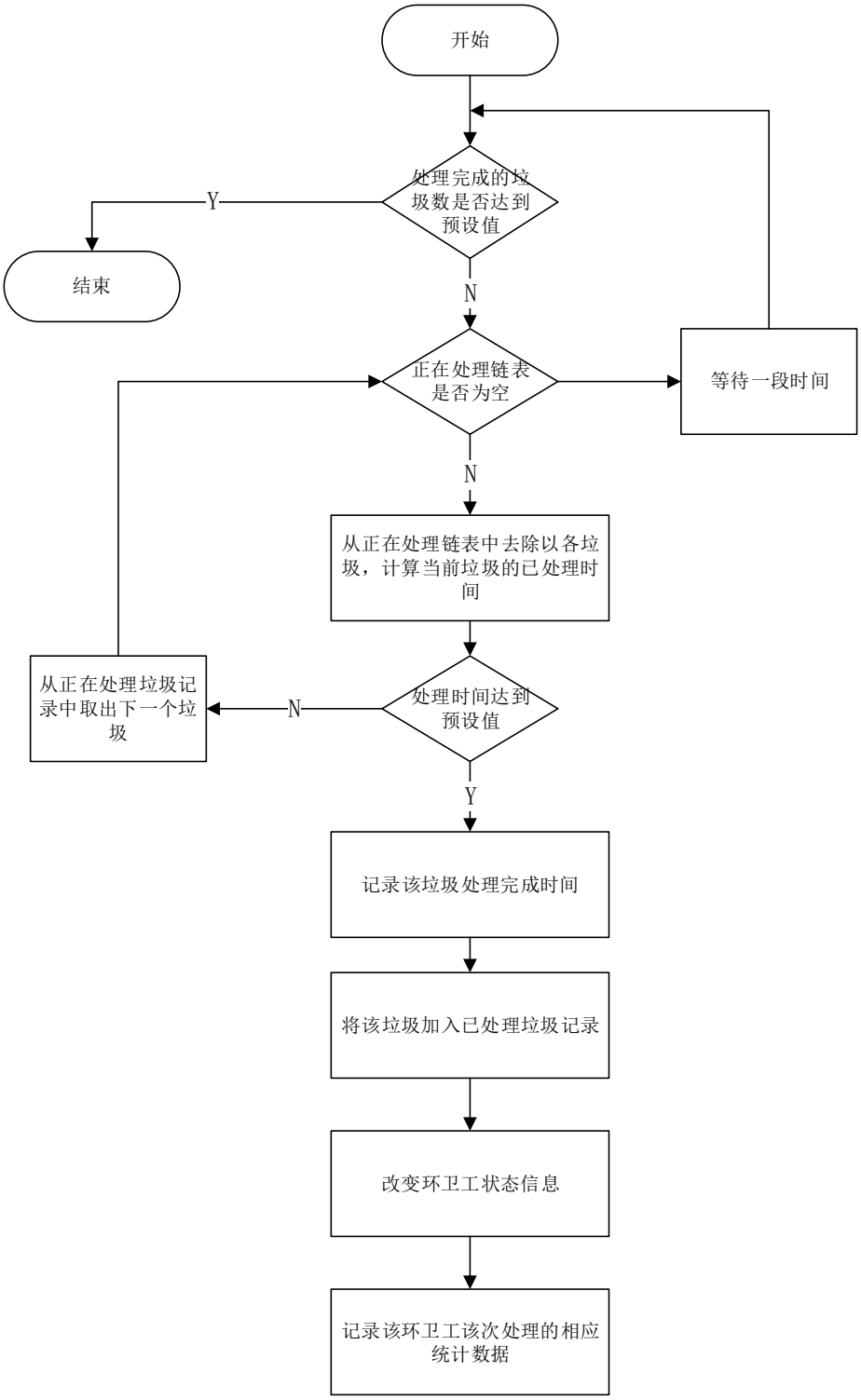

图 3-6 垃圾处理模块流程图

3.3.5 数据采集模块

该模块负责收集其他各个模块产生的各种数据信息，这些数据信息可以存放在相应实体的内部，以实现数据封装的目的。

考虑到能够达到实时获得各模块产生的数据信息，本系统对于数据采集模块，在逻辑设计上与其他模块分离，但在实际的物理实现上采用嵌入式模式。即在每个模块执行完相关操作后，就记录在这一过程中产生的相关统计数据。本模块中主要记录各垃圾的产生时刻，开始处理时刻，处理完成时刻，每个环卫工已经处理的垃圾数量以及 ID。

3.3.6 统计输出模块

该模块在整个系统运作结束后启动，它从数据采集模块获得整个系统在模拟过程中的各种信息，然后利用相关的数学工具计算出相关统计指标提供给用户，用于做下一步的判断处理。在本系统中主要关注的统计指标有每个垃圾的响应时间（开始处理时刻-垃圾产生时刻），每个垃圾的存活时间（处理完成时刻-垃圾产生时刻），环卫工工作强度（处于工作状态时间/整个系统运行时间），环卫工有效工作比率（（处于工作状态时间-移动时间/整个系统运行时间）。

3.4 各模块之间的信息交互机制

因为本模拟仿真系统采用模块化的设计方式，所以会有不同线程模块之间的交互问题。考虑到要使各模块能够实时地并行运行，本系统采取设定全局变量的模式。也就是说，各关键模拟变量在内存中没有拷贝副本，所有关于垃圾的产生，分配，以及调度均是基于共享的全局变量。这样做的好处就是处理简

单，逻辑关系清晰。但缺点就在于在模拟过程中，可能会出现死锁或者读脏数据的情况发生，需要设置相应的并发控制机制来避免这种情况的发生。具体实现方式可见详细设计部分。

3.5 本章小结

本章主要分析了本离散型仿真模拟系统的基本功能需求：产生垃圾，管理环卫工，对待处理垃圾的调度分配，对正在处理垃圾的时间管理，对相关数据的统计分析。并对系统进行了概要设计，将整个系统分为了六大逻辑模块：垃圾产生模块，垃圾调度模块，垃圾处理模块，环卫工管理模块，数据采集模块，统计输出模块。并介绍了各模块之间的数据流动关系以及信息交互方式明确了整个系统的架构模式。在下一章节，将会介绍系统各模块的具体实现细节并进行相应的详细设计。

4 系统详细设计与实现

4.1 系统开发环境与整体实现方法

本离散仿真模拟系统是基于 Windows 操作系统实现的。因为要考虑到该离散模拟系统要满足面向对象的开发要求，并且要有较好的并行性能，所以采用 C++作为该离散仿真系统的开发语言。考虑到对并行开发的需求，采用了 C++11 的标准。最后，系统的开发集成环境（IDE）选择的是 Visual Studio 2017。

系统采用的是将流程模块化，并行运行的设计模式。在逻辑上分为了六大模块：垃圾产生模块，垃圾调度模块，垃圾处理模块，环卫工管理模块，数据采集模块，统计输出模块。但在物理实现上，考虑到模块之间的耦合性问题（比如数据采集模块可以内嵌到其他模块中，垃圾调度模块与环卫工管理模块有着较高的耦合性），并没有完全按照逻辑流程实现，而是做了如下调整：

总共有六个线程：

Main 主线程：负责各数据结构的初始化以及其他子线程的调度协调。

Thread Create Rubbish 线程：该线程负责垃圾的创建。

Thread Allocate Rubbish 线程：负责垃圾的分配。

Thread Treate Rubbish 线程：负责垃圾的处理。

Thread Worker Moving 线程：负责环卫工的移动。

Thread Result Analyze 线程：负责处理最后的各项统计数据。

整个系统先调用主线程，然后主线程负责调用垃圾产生线程，垃圾分配线程，垃圾处理线程，这几个线程是并行运行的，而垃圾分配线程负责调用环卫工移动线程。等到这四个子线程运行完毕后，主线程再调用统计输出线程，等统计输出线程运行结束并输出相关统计数据后，主线程也结束运行，完成整个离散仿真模拟系统的运行。

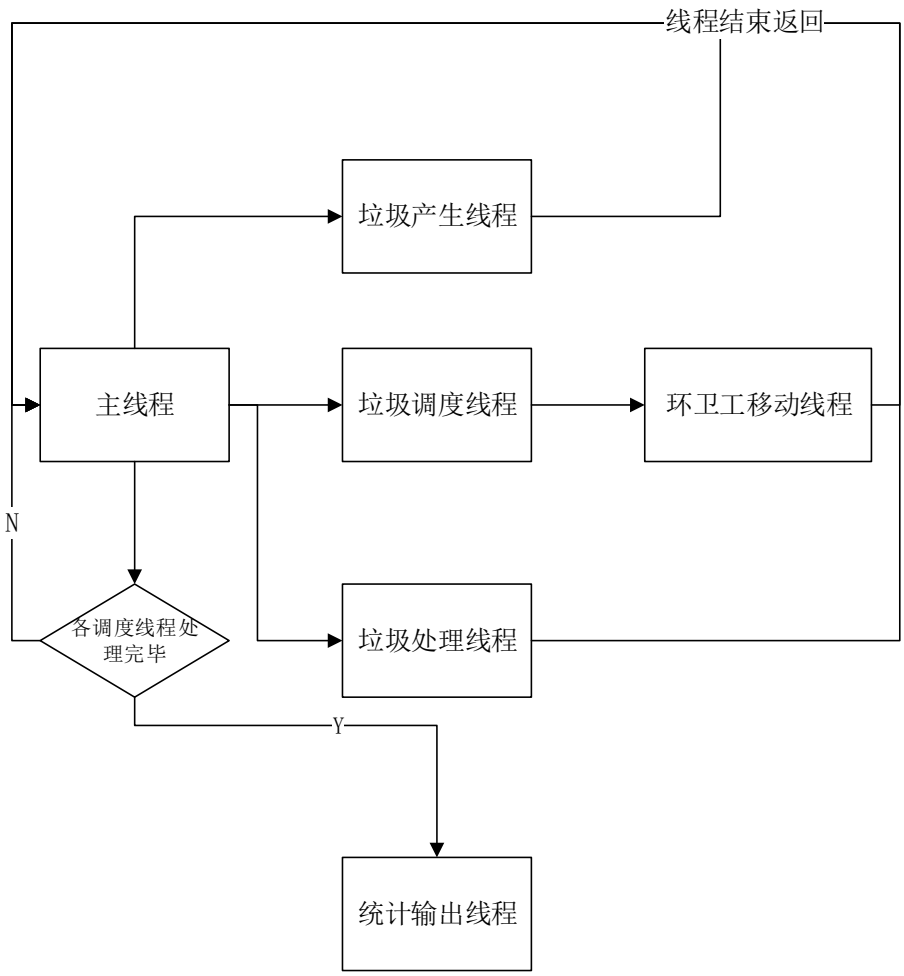

图 4-1 仿真系统线程关系图

4.2 各数据结构设计

本系统中主要存在两类基本的实体类型：垃圾实体，环卫工实体。这两类实体是本模拟系统的原子数据类型，本系统采用了面向对象的设计方式来声明这两类实体的结构。

首先是垃圾实体：考虑到本系统中需要记录处理垃圾过程中的一系列信息，所以对于每个垃圾实体来说，要记录它的 ID 号（从 1 开始，依次累加），产生时刻的位置，产生的时刻，开始处理的时刻，处理完成的时刻。考虑到数据的封装性，需要设置相应的公有接口来实现操作这些数据的目的。具体的设计方案如下所示：

```
class Rubbish
{
public:
    Rubbish();  //默认构造函数
    Rubbish(int x); //构造函数
    const unsigned int GetID()   //返回垃圾 ID
    const clock_t GetCreateTime()      //返回垃圾产生时刻
    const Point GetPosition()  //返回垃圾位置
    void SetProcessTime(clock_t t)    //设置开始处理时刻
    void SetFinishTime(clock_t t)  //设置完成处理时刻
    const clock_t GetProcessTime()//返回开始处理时刻
    const clock_t GetFinishTime()   //返回完成处理时刻
private:
    unsigned int id;    //垃圾 ID
    Point position;     //垃圾位置
    clock_t create_time;    //垃圾产生时刻
    clock_t process_time;   //垃圾开始处理时刻
    clock_t finish_time;    //垃圾处理完成时刻
};
```

其次是环卫工实体：本系统中环卫工主要的任务是报告给调度系统自己的位置与状态，然后接受调度系统的分配，移动到相应位置处理垃圾，处理完成后，再次向调度系统报告自己的位置与状态，等待下一次分配。所以首先要记录每个环卫工的 ID 号（从一开始，依次累加），其次为了能够表示环卫工处理垃圾这一行为，要有工作状态，现在正在处理垃圾，环卫工当前位置，已处理的垃圾数，已处理的垃圾 ID（用链表记录），环卫工每次开始移动的时刻，环卫工处理每个垃圾所用的工作时间（垃圾处理完成时刻-垃圾开始处理时刻），处理垃圾的总共工作时间等信息来记录。

```
class Worker
{
public:
    Worker(Rubbish r);  //构造函数
    const unsigned int GetWorkerID() //返回环卫工 ID
    void SetWorkerID(unsigned int x) //设置环卫工 ID
    const bool GetState()  //返回环卫工工作状态
    void SetState(bool state) //设置环卫工工作状态
    void Allocate(Rubbish r)   //分配垃圾
    Rubbish & FlushRubbish()    //更新垃圾信息
    Rubbish GetRubbish()    //返回正在处理垃圾
    const Point GetRubbishPosition()//返回正在处理垃圾的位置
    const unsigned int GetRubbishID()//返回环卫工正在处理垃圾 ID
    const Point GetPosition()  //返回环卫工位置信息
    void ShowPosition() const;  //显示环卫工位置
    void SetPosition(int x, int y); //设置环卫工位置
    void SetPosition(const Point p);    //设置环卫工位置
    void SetEachWorkTime(clock_t t) //设置每次处理的时间
    const clock_t GetEachWorkTime()//返回每次处理的时间
    const int GetTreatedNum()//返回环卫工已经处理垃圾数
    void FlushTreatedNum()   //刷新环卫工已经处理的垃圾数
    void FlushTreatedID(int x); //刷新环卫工已经处理的垃圾 ID
    void FlushTotalWorkTime()   //刷新环卫工工作总时间
    const clock_t GetTotalWorkTime()   //返回环卫工工作总时间
    void SetStartMovingTime()//设置开始移动环卫工的时间
    const clock_t GetStartMovingTime() //返回开始移动时刻
    void ShowTreatedID();       //显示环卫工处理所有垃圾的 ID
private:
    unsigned int worker_id;     //工人 ID
```

```
    bool busy;  //工作状态
    Rubbish now_rubbish;        //现在正在处理的垃圾
    Point position; //工人当前位置
    int treated_num;    //已处理的垃圾数
    std::list<int> treated_rubbish_id;  //已处理的垃圾 ID
    clock_t start_moving_time;  //环卫工开始移动的时间
    clock_t each_work_time;     //处理每个垃圾的工作时间
    clock_t total_work_time;    //处理垃圾的总共工作时间
}
```

其次，本系统中垃圾的个数与环卫工的个数均为多个。为了方便管理，需要设计相关的辅助数据结构来帮助各线程管理这些资源。考虑到资源会频繁的在各辅助数据结构之间移动，所以采用链表来管理系统运行过程中出现的各资源。

list<Rubbish> untreated list：待处理垃圾链表，负责管理刚产生的垃圾以及待处理的垃圾

list<Rubbish> treating list：正在处理垃圾链表，负责管理正在处理流程中的垃圾

list<Rubbish> treated list：已处理完成垃圾链表，负责管理已经处理完成的垃圾，并记录最后的统计信息

list<Worker> worker array：处于静止状态的环卫工链表，负责管理处于空闲状态和正在处理垃圾状态的环卫工

list<Worker> worker array moving：处于运动状态的环卫工链表，负责管理正在运动状态的环卫工

为了实现系统中各线程资源与信息的交互，各链表均采用全局变量（生存期为整个进程，并且具有外部链接性）：

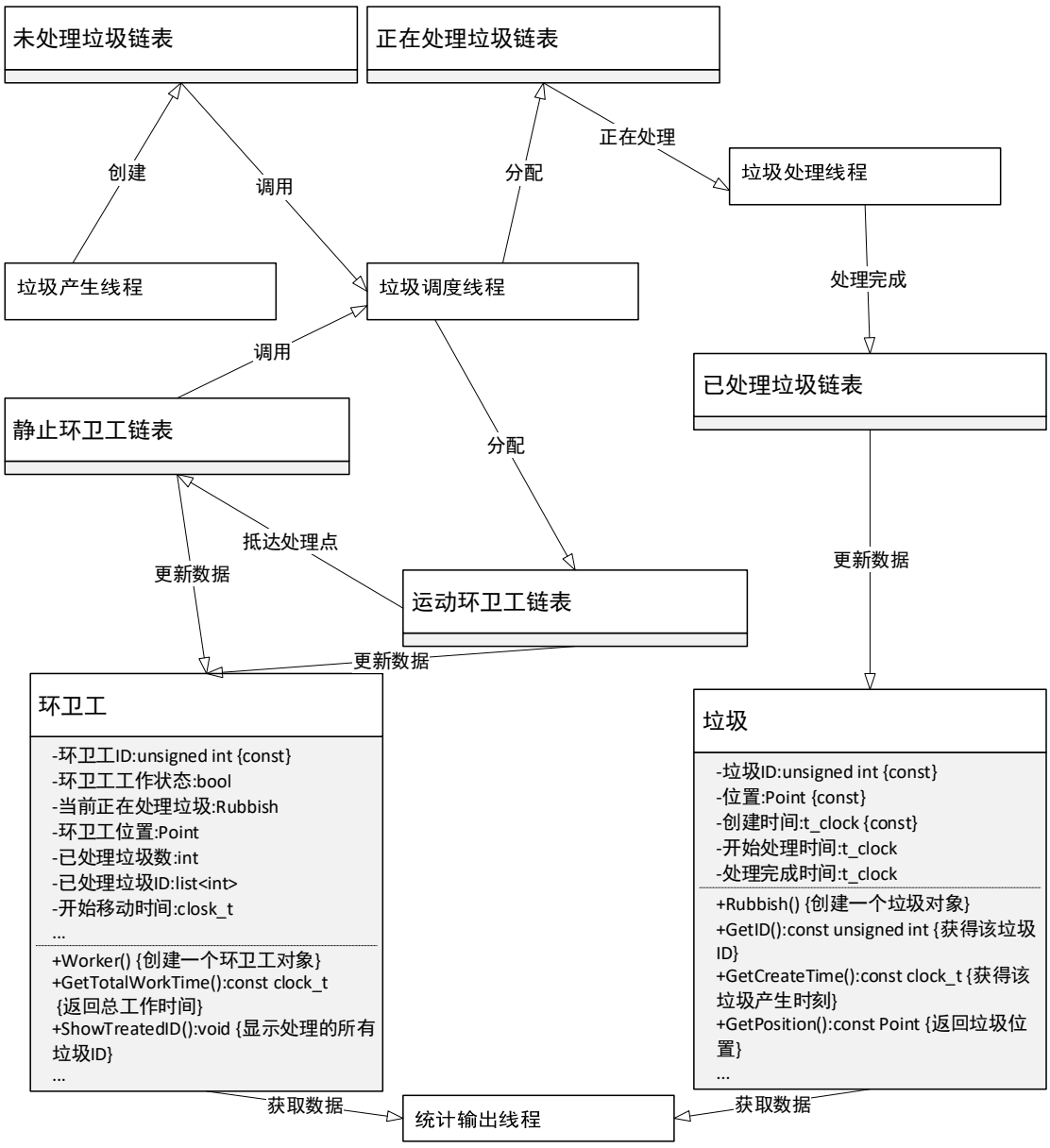

图 4-2 数据结构关系图

4.3 各线程模块实现方法

4.3.1 主线程模块实现

在本模块主要负责初始化五个链表结构，并调用环卫工初始化函数，设定

该系统的环卫工状态：

InitWorker()；

因为本模拟系统假设的地图大小为 90*90 的正方形，所以总共初始化了九个环卫工，这九个环卫工初始位置依次是(15, 15)，(45, 15)，(75, 15)，(15, 45)，(45, 45)，(75, 45)，(15, 75)，(45, 75)，(75, 75)，这九个点分别是该地图的九个子中心点。整个函数的流程图如下：

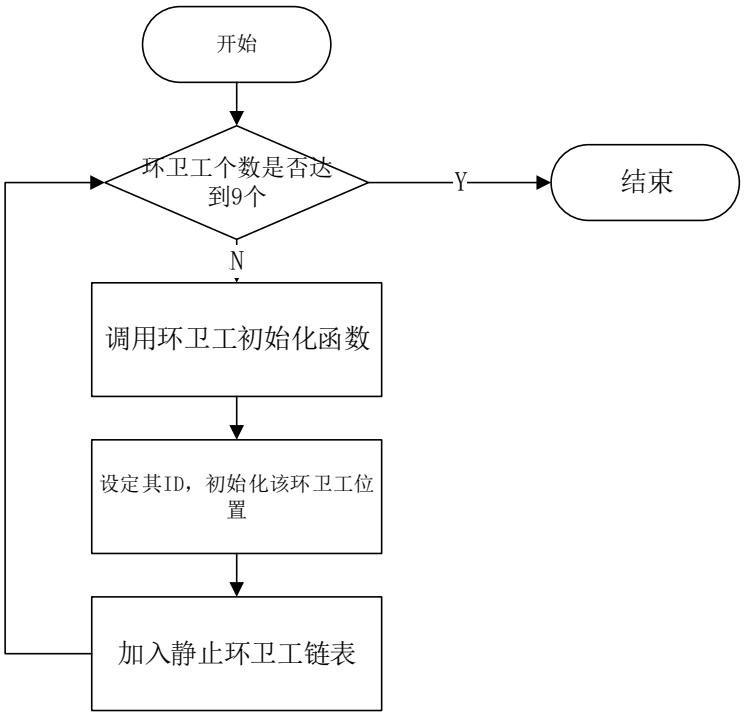

图 4-3 初始化函数流程图

完成初始化后，主线程调用垃圾产生线程，垃圾分配线程，垃圾处理线程，交出系统控制权并处于阻塞状态，等到上面三个线程运行完毕后，再调用统计分析线程，再次交出控制权并处于阻塞状态，在统计分析线程运行完毕后，重新获得控制权，结束仿真过程。这里的线程调用使用的是 C++的<Thread>标准库。整个主线程流程示意图如下：

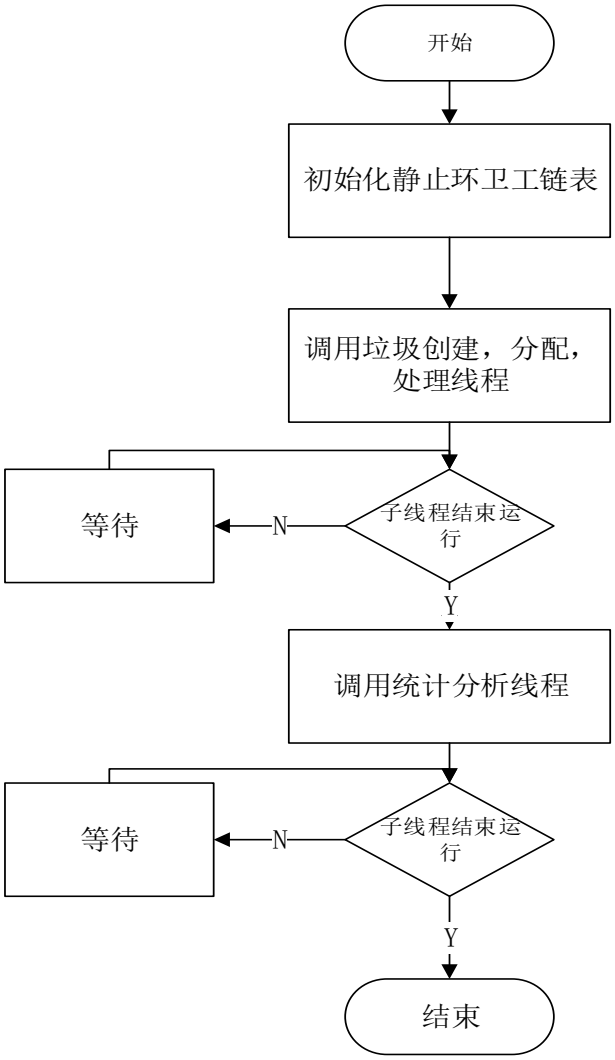

图 4-4 主线程流程图

4.3.2 垃圾产生线程模块实现

该线程负责随机产生垃圾并报告垃圾相关信息，主要输出操作对象是待处理垃圾链表。考虑到现实生活中垃圾产生的不确定性，本模拟系统不可能完全刻画出这种随机性，所以利用统计学规律来近似的模拟这种随机性，这里按均匀分布（每个地方等可能概率）来产生垃圾，其中每隔 1.5s 产生一个垃圾（注：这里垃圾的产生间隔可以在系统初始化前人为设定）。具体是通过<stdlib.h>库中的随机数产生函数实现随机产生垃圾的目的：

srand((unsigned)time(NULL));

position.x = rand() % 90;　　//水平范围[0,89]

position.y = rand() % 90;　　//垂直范围[0,89]

考虑到在现实生活中垃圾产生后被人发现并报告给系统是需要有一定时间的，为了近似这种效果，每隔一秒钟，该垃圾才会被加入待处理垃圾链表（注：这里的发现时间同样可以在系统初始化前人为设定）。所以，在垃圾创建线程模块总共有两处延迟，具体实现方法是通过调用以下函数：

std::this_thread::sleep_for(std::chrono::milliseconds(1000));　　//平均过 1 秒，垃圾就会被发现，并报告

std::this_thread::sleep_for(std::chrono::milliseconds(1500));　//程序休眠 1.5 秒，产生下一个垃圾

最后，因为在现实生活中，垃圾的产生是无限的。如果让该线程无限的运行下去，虽然技术上是可行的，但会浪费大量系统资源。从性能与数据有效性的角度来看，让该线程产生一定数量的垃圾也是可行的（节约了系统资源并且得到的数据结果同样具有参考价值）。所以本系统设定了产生垃圾总量的阈值（这里设定为了 20）：

```
const unsigned int TOTAL_RUBBISH = 20;　　//需要处理的垃圾总数，单位个
```

当产生的垃圾总数达到阈值后，就会结束运行，把控制权交回给主调用线程。

4.3.3 垃圾分配线程模块实现

垃圾分配线程主要的输入操作对象是静止状态环卫工链表和待处理垃圾链表，主要的输出操作对象是移动状态环卫工链表，而在本模块中主要负责有三个任务：

1．检查已处理完成的垃圾数是否达到阈值，如果没有，继续进行；反之结束线程，把控制权交回给主调用线程。

2．执行相关的调度算法，实现垃圾的调度分配。如果要选用不同的调度算法，只需要更改这一模块就可以达到目的，而不需要更改整个模拟程序其它的地方，这也进一步的提高了该系统的可扩展性。

3．调用环卫工移动线程，实现环卫工移动这一物理特征。为了只调用一次环卫工移动子线程，设立 moving_thread_flag 标志变量用来记录

```
static bool moving_thread_flag = false;
                    if (moving_thread_flag == false)
                    {
                        std::thread thread_worker_moving(work_moving);
                        thread_worker_moving.detach();
                        moving_thread_flag = true;
                    }
```

（一）顺序调度算法实现：

该算法是通过取出待处理垃圾链表中的第一个垃圾，然后顺序遍历静止环卫工链表，找到第一个处于空闲状态的环卫工，将该垃圾分配给该环卫工。该分配算法的优势在于分配简单，整个算法的时间复杂度为 O（n），但缺点就是可能将一个垃圾分配给了一个处于较远位置的环卫工，导致垃圾的处理时间变长。

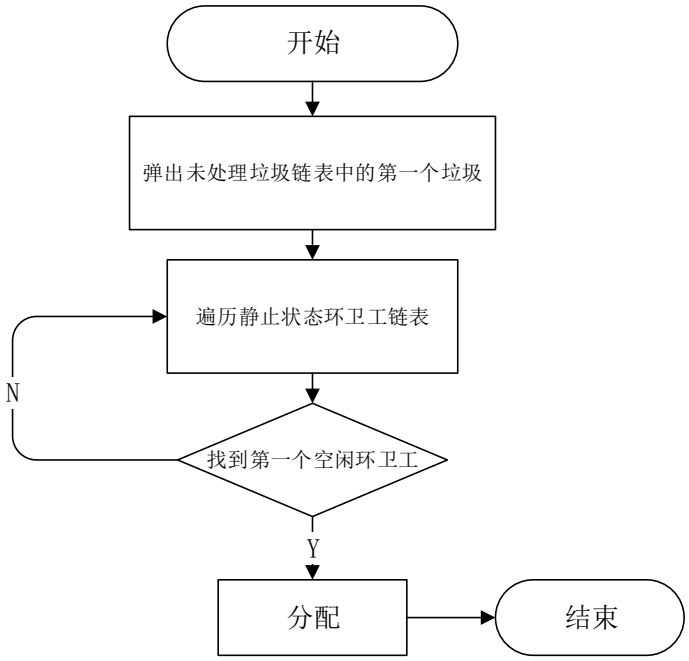

图 4-5 顺序调度算法流程图

从待处理垃圾链表中弹出第一个待处理垃圾是通过以下语句实现的：

```
Rubbish temp = untreated_list.front();      //取出一个未处理垃圾
untreated_list.pop_front();                 //将该垃圾从未处理垃圾链表中删除
```

（二）随机调度算法实现：

该算法先从待处理垃圾链表中取出一个垃圾，然后从静止状态环卫工链表中选取一个处于空闲状态的环卫工，将垃圾分配给该环卫工。整体的流程与顺序调度算法基本一致，但要考虑随机调度的实现，这里的随机选取是通过判断产生的随机数的奇偶性来确定是否选取，所以说，每个环卫工均有 50%的可能性会被选中。

```
srand((unsigned)time(NULL));
if ((*it).GetState() == false && rand() % 2 == 0)
    {
        /* . . . */
    }
```

（三）区块调度算法实现：

该算法将整个大区域划分为了几个小区域，每个环卫工负责其中的一个小区域。因为本模拟系统中的地图为 90*90 的矩形区域，共有九个环卫工。所以在该算法中，每个环卫工负责的区域大小是 30*30，具体规划图如下：

[0, 30) [0, 30)	[30, 60) [0, 30)	[60, 90) [0, 30)
[0, 30) [30, 60)	[30, 60) [30, 60)	[60, 90) [30, 60)
[0, 30) [60, 90)	[30, 60) [60, 90)	[60, 90) [60, 90)

（注：第一行为环卫工横坐标的覆盖范围，第二行为环卫工纵坐标的覆盖范围）

为了实现环卫工的区块调度，需要将环卫工的工作覆盖区域添加到环卫工实体中，所以需要修改相应的环卫工数据结构，而每个环卫工具体的工作范围是在环卫工初始化函数中实现的。

该算法的流程图如下所示：

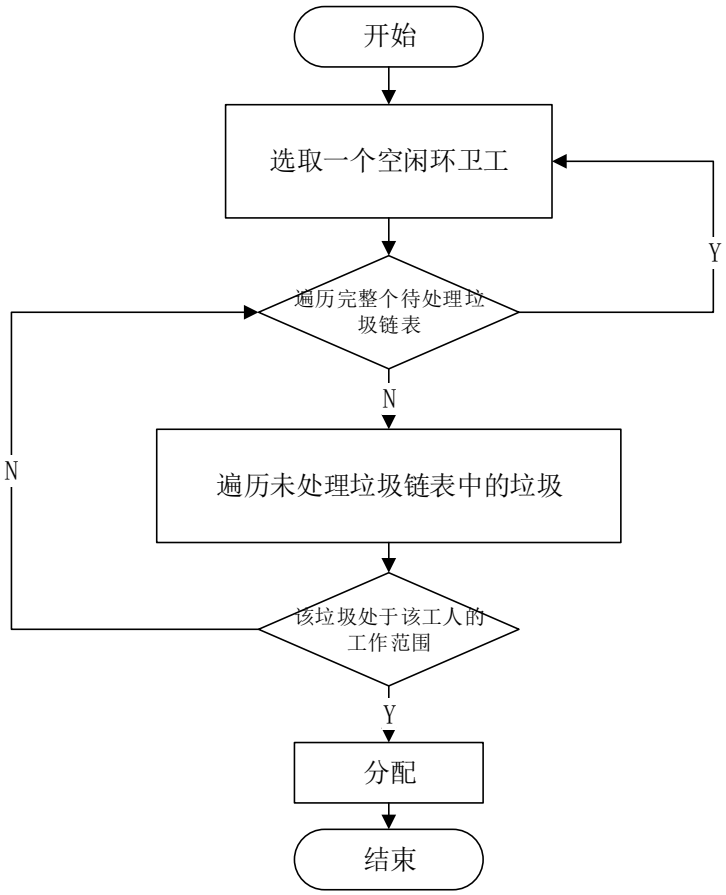

图 4-6 区块调度算法流程图

（四）最短距离调度算法实现：

该算法是通过计算待处理垃圾离每一个处于空闲状态环卫工的距离，找到离它最近的环卫工，实现调度分配。

考虑到计算机的像素显示均为整型值，所以这里的距离计算采用的是绝度值距离。

```
int abs_distance(const Point a,const Point b)
{
    return abs(a.GetX() - b.GetX()) + abs(a.GetY() - b.GetY());
}
```

该算法的流程图如下所示：

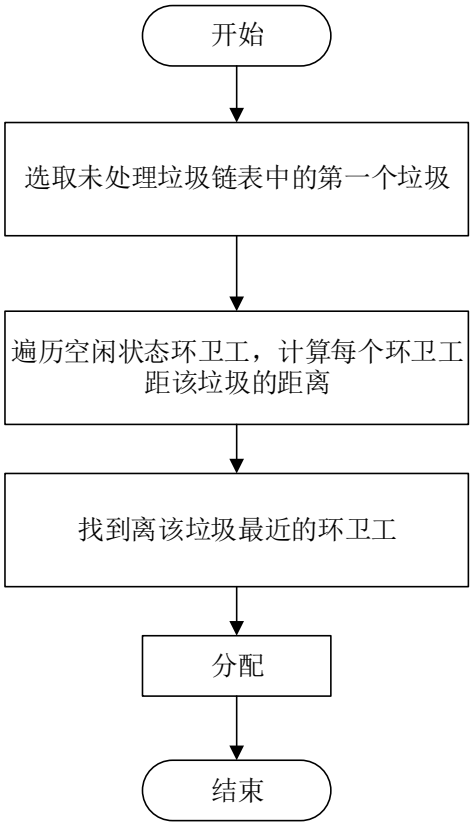

图 4-7 最短距离调度算法流程图

在该算法中，寻找最近的环卫工是通过函数实现的，该函数接收一个垃圾为输入参数，并返回离该垃圾最近的环卫工的迭代器：

```
std::list<Worker>::iterator FindMinWorker(Rubbish t)
{
    std::list<Worker>::iterator min_it;
    std::list<Worker>::iterator it;
    int dis = 5000;
    for (it = worker_array.begin(); it != worker_array.end(); it++)
    {
        if (abs_distance(t.GetPosition(), (*it).GetPosition()) <= dis &&
(*it).GetState() == false)
        {
```

```
            dis = abs_distance(t.GetPosition(), (*it).GetPosition());
            min_it = it;
        }
}
    return min_it;
}
```

本模拟仿真系统共采用的如上四个分配算法（顺序调度算法，随机调度算法，区块调度算法，最短距离调度算法）均采用的嵌入式模块的设计模式，针对不同的调度方式，只需要设计相应的调度算法，而不需要对整个系统做整体的改动，这样会极大地提高该系统的可扩展性，可以按照不同的仿真要求设计不同的调度算法模块，然后通过接口嵌入本仿真系统即可。

4.3.4 环卫工移动线程模块实现

本模块的主要输入操作对象是移动状态环卫工链表，主要的输出操作对象是静止状态环卫工链表和正在处理的垃圾链表。为了反映出环卫工处理垃圾的速度取决于他离该垃圾的位置大小这一客观因素，需要通过环卫工移动模块来实现。

该模块由垃圾分配线程调用，调用成功以后，使用 detach（）函数与垃圾分配线程解除绑定关系，成为独立线程。

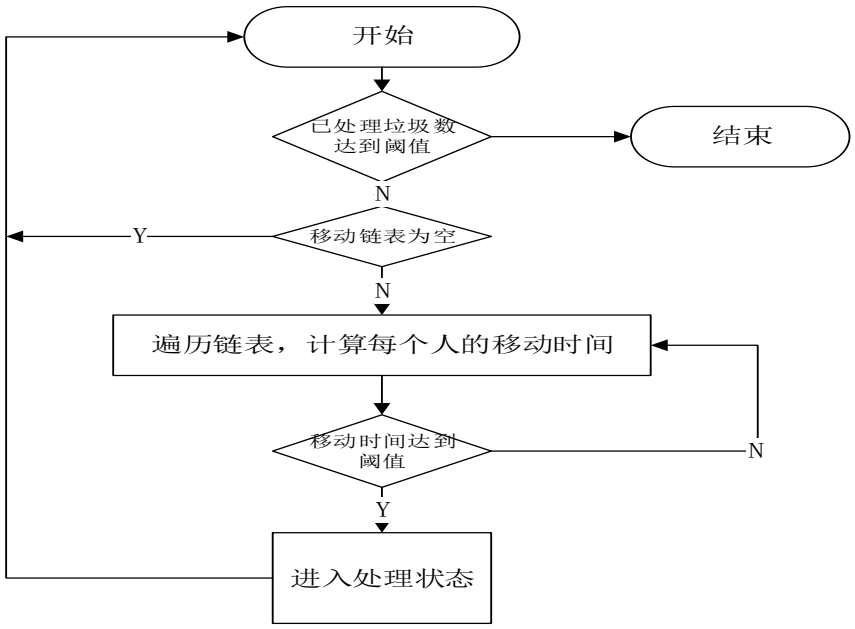

图 4-8 环卫工移动线程流程图

在环卫工移动模块中，本系统的遍历模式采用的是顺序遍历，因为环卫工链表是线性结构，所以采用顺序遍历的时间复杂度较低。

在本线程模块中的核心部分是移动时间的计算。本系统中采用的是环卫工移动速率（移动速率可以在系统初始化前人为设定，移动速度的单位是 10 米/秒）乘以环卫工当前位置与垃圾位置之差来计算移动时间。

```
if (clock() - (*it).GetStartMovingTime() >= (100 *
abs_distance((*it).GetPosition(), (*it).GetRubbishPosition())))
{
    /* . . . */
}

int abs_distance(const Point a,const Point b)
{
    return abs(a.GetX() - b.GetX()) + abs(a.GetY() - b.GetY());
}
```

每当一个环卫工达到他所要处理垃圾的位置点时，便输出相关信息，并将该环卫工从移动状态转化为静止状态，即将其从环卫工移动链表中取出，记录下该环卫工开始处理垃圾的时刻，将每次处理时间重置，用于记录该次重置时间，然后加入到静止环卫工链表尾端，完成状态的转换工作。注意这时环卫工的工作状态处于工作状态。

```
worker_array.push_back(*it);

it = worker_array_moving.erase(it);
```

4.3.5 垃圾处理线程模块实现

本线程模块是整个处理流程中的最后一个模块，该模块的主要的输入操作对象是正在处理垃圾链表和静止环卫工链表，而输出操作对象是已处理完成垃圾链表。

现实生活中不同的环卫工处理不同的垃圾时，由于会受到许多不可测因素的影响，会导致处理时间有所差异，但从宏观角度来说，根据中心极限定理，垃圾的平均处理时间也是服从正态分布的，所以本系统通过设定垃圾的平均时间为常量（该常量可以在系统初始化前人为设定），实现垃圾处理完成与否的标准。

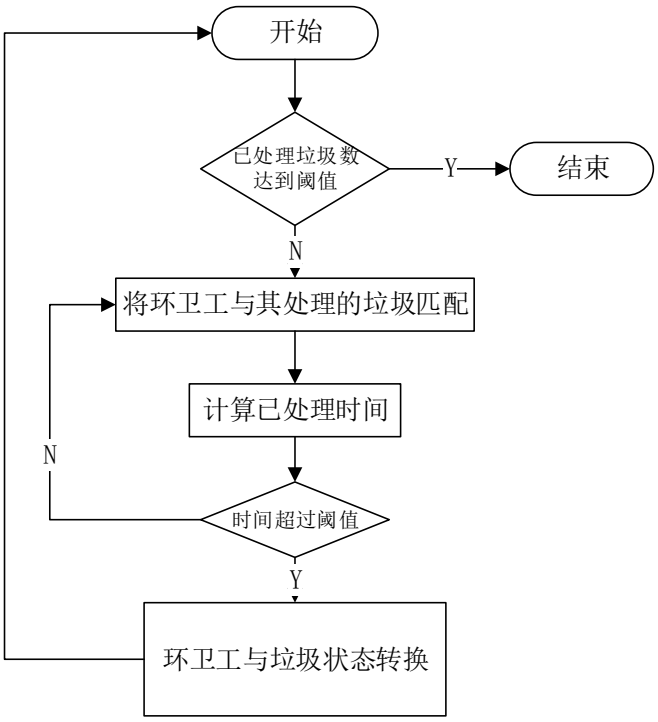

图 4-9 垃圾处理线程流程图

因为环卫工与其处理的垃圾分别在两个链表中，所以需要进行相应的匹配操作，这里采用将正在处理的垃圾 ID 绑定到相应环卫工对象中的方法实现绑定。这里处理时间的计算是通过以下表达式计算获得：

```
if (clock() - (*it).GetProcessTime() >= TIME_NEED_TO_FINISH * CLOCKS_PER_SEC)
```

对于已经处理完成的垃圾，通过以下操作实现对垃圾状态和环卫工状态的转换：

```
(*worker_it).SetState(false);            //工人状态由忙变为空闲
(*worker_it).FlushTreatedNum();          //将该工人处理的垃圾数加 1
(*worker_it).FlushTreatedID((*it).GetID()); //将处理的垃圾 id 加入该工人已处理
垃圾 id 链表
(*it).SetFinishTime(clock());            //记录下该垃圾完成时刻
(*worker_it).SetEachWorkTime((*it).GetFinishTime() -
(*worker_it).GetEachWorkTime());    //记录该工人处理这个垃圾的时间
(*worker_it).FlushTotalWorkTime();          //更新这个工人工作总时间
Rubbish temp = (*it);                    //将该垃圾从正在处理垃圾链表中取出
treated_list.push_back(temp);            //加入已处理完成垃圾链表
it = treating_list.erase(it);      //将该垃圾从正在处理垃圾链表中删除
```

至此，本模拟仿真系统的具体运作部分已经实现完成。

4.3.6 统计输出线程模块实现

本线程模块负责收集并整理整个系统产生的各类信息数据。该模块的主要输入操作对象是已处理完成垃圾链表和静止状态环卫工链表。而整个数据的处理又分为两大部分，第一部分是实时数据的输出，用于使用者实时监测垃圾处理情况；第二部分是在系统模拟完成后，对整个系统模拟过程中产生的所有数据进行统计分析。

实时数据的输出：在整个模拟过程中，需要系统能够给用户提供实时的信息，方便调用者监控整个模拟仿真过程是否稳定。这一部分与垃圾产生线程，垃圾分配线程，垃圾处理线程，环卫工移动线程的耦合度都比较高。所以本系统采用的是嵌入式的设计方式，即每当各线程完成相关操作后，就会输出相关的数据信息提供给调用者。

——当一个垃圾产生时，输出下列信息：

“A rubbish has been created: ID (该垃圾 ID) create time:（该垃圾产生时间）seconds。Position(（该垃圾位置横坐标），（该垃圾位置纵坐标）)。”

——当一个垃圾被发现并提交给系统时，输出下列信息：

"This rubbish(ID: (该垃圾 ID)) has been found and reported。"

——当一个垃圾被分配给相应的环卫工时，输出下列信息：

"Worker[（该环卫工 ID）] moving to （分配给该环卫工的垃圾 ID） rubbish。"

——当一个已被分配的垃圾开始被环卫工处理的时候，输出下列信息：

"Worker[（该环卫工 ID）] treating number （分配给该环卫工的垃圾 ID） rubbish。"

" Worker Position:（环卫工位置信息）。"

——当该垃圾被分配给他的环卫工处理完成时，输出下列信息：

"Number （该垃圾 ID） rubbish has been treated。"

" Finish time: （垃圾完成时刻）。"

除此之外，当各线程开始运行或者运行结束后，也会输出关于进程启动和结束的信息提供给调用者用以监视。

对最后所有信息数据的统计输出：当整个模拟仿真结束后，该部分将会作为独立的进程模块从相应的已处理完成垃圾链表和静止状态环卫工链表中提取收集到的相关信息，并利用相关的数学工具转换为统计学指标，提供该调用者，用以作参考。

对于垃圾实体来说，总共有如下八项指标：

1.每个垃圾被发现并报告的时刻

2.每个垃圾开始处理的时刻

3.每个垃圾处理时间(该垃圾完成时刻-该垃圾开始处理时刻)

4.每个垃圾响应时间(该垃圾开始处理时刻-该垃圾发现时刻)

5.每个垃圾存活时间(该垃圾完成时刻-该垃圾发现时刻)

6.平均总响应时间(平均开始处理时刻-平均发现时刻)

7.平均总存活时间(平均完成时刻-平均发现时刻)

8.平均总处理时间(平均完成时刻-平均开始处理时刻)

对于环卫工来说，总共有如下四项指标：

1.每个工人处理垃圾的总个数

2.每个工人处理的垃圾 ID

3.每个工人的工作总时间

4.每个工人的工作强度（工作总时间 / 系统运行时间）

在数据采集模块，收集到的时间数据的单位是 clock_t，这是 C++的内置时间类型，所以在统计输出的时候，要将对相应的内置时间类型数据做相应的转换，才可以输出给用户。具体在不同的机器上，该转换关系可能有所不同，在我的开发环境下，是 1 秒 = 1000 clock_t（Windows10 + Corei5）。当该系统移植到不同的环境下时，系统会自动设置相应的转换变量 CLOCKS_PER_SEC。

4.4 各线程模块的并发控制

本仿真模拟系统是一个多线程模拟系统，而且不同的线程都是对同一组数据进行操作。所以需要在各线程之间设置相应的并发控制机制。

本系统采用了设置互斥变量的方式来保证数据操作的安全，这里使用的是 C++的 mutex 标准库。

```
mtx.lock();
/* ... */
mtx.unlock();
```

同时为了充分发挥各线程之间的并行性，要尽可能的缩短上锁的代码量，所以对于以下几种情况，系统使用 lock 技术进行并发控制：

在各实体进行状态转换的过程中，需要修改一系列的数据，所以要保证在修改期间，其他线程不会来抢夺资源，否则会出现读脏数据的问题。

在实时数据输出的时候，整个仿真系统共享一个输出流对象资源（cout），这里也需要设置互斥变量来保证输出不会出现格式错乱的情况。

在并发控制机制设置中，还要注意关于链表指针更新的问题。当一个线程在遍历过程中，其他线程可能会对该链表的数据项进行相应的增删操作，这会导致该线程的指针在解引用时出现异常，所以需要在每次遍历的过程中更新该链表指针，避免指针无法解引用的情况出现。

4.5 文件组织

考虑到为了便于后期的维护与继续开发扩展，本系统将开发过程按模块封装在了不同文件中，下面是各文件的引用关系图：

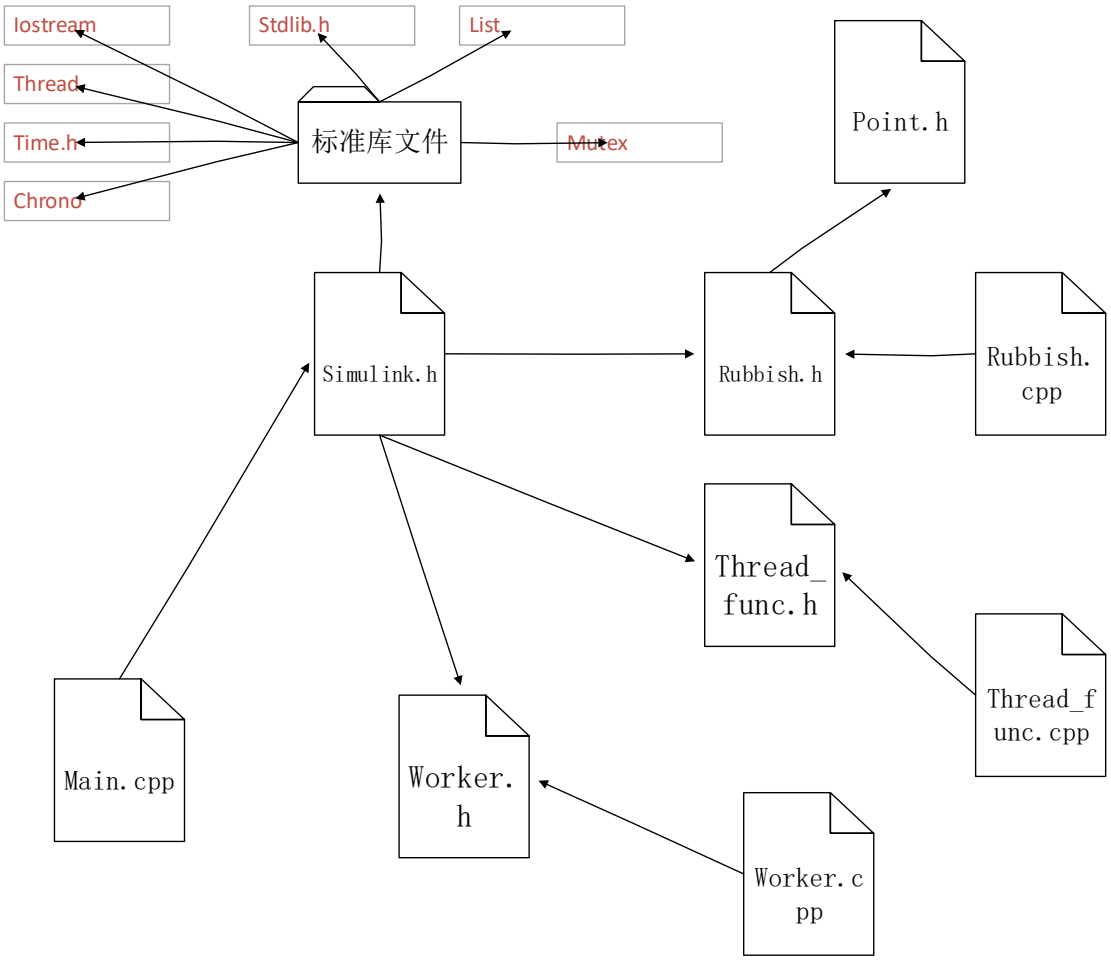

图 4-10 文件引用关系图

各模块之间的内容是相互独立的，并且使用头文件作为各模块之间的接口，后期可以方便的进行修改扩展与调试。

4.6 本章小结

本章的主要内容是给出垃圾仿真模拟系统的具体实现方式，基于前一章的概要设计，本章经一步细化，详细介绍了主线程，垃圾产生线程，垃圾调分配线程，环卫工移动线程，垃圾处理线程，统计输出线程等六大模块的实现，以及各线程之间的调用关系和并发控制机制的设计，最后还介绍了个源代码文件之间的组织关系。而具体的系统模拟测试以及各算法之间的差异比较将会在系统测试结果章节介绍。

5 系统测试结果分析

5.1 测试参数设置

表 5-1 系统参数

系统总共处理的垃圾数	20 个
每个垃圾被发现所需要的延时	1.5 秒
环卫工移动速度（有代步工具）	10 米/秒
处理每个垃圾所需要的时间	5 秒

表 5-2 软硬件参数

软硬件类型	属性
操作系统	Windows10
CPU 型号	Inter(R) Core i5-2450M

5.2 调度测试

在系统运行过程中，会实时产生每个垃圾实体与环卫工实体的状态信息，当垃圾产生，垃圾被发现，垃圾调度分配，环卫工移动，环卫工开始处理垃圾以及垃圾处理完成时，均会产生相应的输出信息，以下是在最短距离调度模式下的实时运行结果：

C:\WINDOWS\system32\cmd.exe

```
A rubbish has been created: ID 1 create_time:0.001 seconds.
Position(41,25)

This rubbish(ID:1) has been found and reported

worker[2] moving to 1 rubbish.

worker[2] treating number 1 rubbish.
worker Position:(41,25)

A rubbish has been created: ID 2 create_time:2.519 seconds.
Position(48,4)

This rubbish(ID:2) has been found and reported

worker[3] moving to 2 rubbish.

A rubbish has been created: ID 3 create_time:5.037 seconds.
Position(57,21)

This rubbish(ID:3) has been found and reported

worker[5] moving to 3 rubbish.

worker[3] treating number 2 rubbish.
worker Position:(48,4)

number 1 rubbish has been treated.
finish time 7.412
```

当系统运行完毕后，会调用统计输出线程产生相关的统计信息，该统计信息主要分为两大部分：第一部分，有关垃圾实体的统计信息，第二部分，有关环卫工实体的统计信息。以下是在最短距离调度模式下的统计运行结果：

```
find time: 0.001 seconds
start process time: 2.412 seconds
process time: 5 seconds
responsive time: 2.411 seconds
live time: 7.411 seconds


2 rubbish:
find time: 2.519 seconds
start process time: 7.331 seconds
process time: 5 seconds
responsive time: 4.812 seconds
live time: 9.812 seconds


3 rubbish:
find time: 5.037 seconds
start process time: 9.648 seconds
process time: 5 seconds
responsive time: 4.611 seconds
live time: 9.611 seconds


4 rubbish:
find time: 7.555 seconds
start process time: 12.57 seconds
process time: 5 seconds
responsive time: 5.015 seconds
live time: 10.015 seconds
```

```
worker 4 has treated 2 rubbish
treated rubbish id: ->8->18
work total time: 10 seconds
work strength: 17.1356 %


worker 5 has treated 4 rubbish
treated rubbish id: ->3->9->13->17
work total time: 20 seconds
work strength: 34.2712 %


worker 6 has treated 3 rubbish
treated rubbish id: ->5->14->20
work total time: 15 seconds
work strength: 25.7034 %


worker 7 has treated 0 rubbish
treated rubbish id: worker 7 has not treated any rubbish
work total time: 0 seconds
work strength: 0 %


worker 8 has treated 0 rubbish
treated rubbish id: worker 8 has not treated any rubbish
work total time: 0 seconds
work strength: 0 %
```

5.3 各算法运行结果对比

本节主要针对本仿真系统中设计的四个调度分配算法进行效率分析，具体的模拟结果如下表格所示：

（1）垃圾存留时间对比（单位:秒）

垃圾实体	顺序调度	随机调度	区块调度	最短距离调度
1	12.82	15.982	7.607	7.411
2	8.75	14.463	6.019	9.812
3	10.336	20.206	9.672	9.611
4	13.67	12.051	27.492	10.105
5	9.86	10.457	25.425	8.144
6	11.604	9.679	29.993	9.212
7	17.966	19.071	11.892	8.113
8	14.42	13.437	6.613	11.313

9	16.03	14.876	13.944	10.513
10	11.481	13.463	8.219	10.016
…	…	…	…	…
18	16.021	11.256	7.323	9.914
19	15.839	17.947	10.458	8.414
20	7.706	14.17	15.729	10.514

数据结果分析：

从以上数据对比中可以发现随机调度算法中垃圾的存留时间最长，其次是顺序调度算法，再次是区块调度算法，而最短距离调度算法中垃圾的存留时间最短。而除了平均存留时间的对比之外，还可以看出区块调度算法中垃圾的存留时间有较大的方差。

（2）环卫工处理垃圾 ID 与工作强度对比

环卫工实体		顺序调度	随机调度	区块调度	最短距离调度
1	处理垃圾 ID	->1->11->19	->3->17	->10->12	->7->10->19
	工作强度	23.7839 %	12.9049 %	11.6836 %	25.7034 %
2	处理垃圾 ID	->2->10->18	->2->11->20	->14->16	->1->4->11->15
	工作强度	23.7839 %	19.3573 %	11.6836 %	34.2712 %
3	处理垃圾 ID	->3->12	->4->14	->8->18	->2->6->12
	工作强度	15.8559 %	12.9049 %	11.6836 %	25.7034 %
4	处理垃圾 ID	->4->14	->5->12	->11->13	->8->18
	工作强度	15.8559 %	12.9049 %	11.6836 %	17.1356 %
5	处理垃圾 ID	->5->13	->6->15	->2->15	->3->9->13->17
	工作强度	15.8559 %	12.9049 %	11.6836 %	34.2712 %
6	处理垃圾 ID	->6->15	->8->16	->4->6->7->9->17->19->20	->5->14->20
	工作强度	15.8559 %	12.9049 %	64.4036 %	25.7034 %
7	处理垃圾 ID	->7->17	->9->13->19	NULL	NULL

	工作强度	15.8591 %	19.3573 %	0	0
8	处理垃圾 ID	->8->16	->1->10	->1->3	NULL
	工作强度	15.8591 %	12.9049 %	11.6836 %	0
9	处理垃圾 ID	->9->20	->7->18	->5	->16
	工作强度	15.8591 %	12.9049 %	26.9961 %	8.56781 %

数据结果分析：

在环卫工实体的数据中，可以看出在顺序调度算法中，每个环卫工处理的垃圾 ID 基本是有序的，而且每个环卫工的工作强度大致相同。

在随机调度算法中每个环卫工处理的垃圾 ID 是无序的，但由于每个环卫工都是等概率的处理垃圾，所以工作强度也大致相同。

在区块调度算法中每个环卫工处理的垃圾均是处于他所在的工作范围之内，但由于不同区域垃圾的产生频数不同，导致不同区域的环卫工工作强度差异较大，有的达到了 64%，而有的却没有垃圾可处理，导致工作强度为 0。

在最短距离调度算法中每个环卫工处理的垃圾均是在他工作路径中离他最近的垃圾，所以其所处理的垃圾 ID 均是无序的。由于垃圾产生位置的随机性，也会导致不同环卫工工作强度的差异，但这种波动性要小于区块调度算法，虽然仍有环卫工的工作强度为 0，但不会出现工作强度大于 50%的情况。

5.4 各算法优劣势对比

1.顺序调度算法：垃圾存留时间较长，但波动性较小。每个环卫工工作强度一定，但灵活性较低。

2.随机调度算法：垃圾存留时间较长，波动性较大。每个环卫工工作强度一定，但有较大的不确定性。

3.区块调度算法：垃圾存留时间较短，每个垃圾需要等待环卫工处理完上一个垃圾后，才可以开始被处理，导致波动性较大。由于垃圾产生位置的随机性，导致每个环卫工工作强度有很大的波动性。

4．最短距离调度算法：垃圾存留时间短，波动性较小。环卫工工作强度也有一定波动性，但情况要优于区块调度算法。

5.5 本章小结

本部分基于之前设计的模拟仿真系统进行了测试分析，并分析了不同算法下整个系统运作效率的差异，总结了不同调度算法的优劣势。本测试只是考虑了垃圾处理中的主要流程，对于许多具体的物理细节（不同垃圾的不同特性，不同环卫工的工作能力差异等…）还没有实现。在今后的研究中，我会逐步完善该模拟系统中的其他物理细节,并进一步研究探索更有效率的调度算法。

6 总结与展望

6.1 工作总结

本次计算机双学位的毕业设计是我本科阶段的第一份毕业设计，考虑到我主专业（物流管理）的特点，在导师的建议下，选择了关于调度仿真方面的课题。本次毕业设计的过程可以大致分为三个阶段：

第一阶段通过查阅文献以及相关资料，充分了解目前街道垃圾的主要处理流程，并通过学习关于计算机离散仿真方面的知识，完成逻辑流程图和数据流图的设计。

第二阶段在导师的指导与帮助下，完成整个仿真模拟框架的设计搭建并进行相关样例测试。整个框架按进程模块被划分为了主线程模块，垃圾产生模块，垃圾调度模块，垃圾处理模块，环卫工移动模块，统计输出模块等六大模块，各模块通过接口耦合，以实现整个对街道垃圾处理流程的模拟仿真。

第三阶段设计四种不同的调度分配解决方案算法，并对比不同方案之间的性能差异。本系统总共提供了四种调度解决方案：顺序调度分配，随机调度分配，按区块划分调度分配，最短距离调度分配。其中后两种解决方案在响应性与灵活性上都有着较好的表现。

为了达到更好的仿真效果以及今后更好的维护与扩展，本系统采用了面向对象模块化的，多线程开发技术，并通过互斥变量的锁机制实现各线程资源的有序使用，解决了死锁问题出现。

在构建框架的过程中，由于开发时间有限，本模拟仿真系统框架是基于相关的假设实现的，虽然完成了相关的基本功能要求，但不能完全反映现实情况中的所有细节，在今后的工作中，可以考虑逐渐放宽相关假设。除此之外，在系统可靠性，可用性，资源利用率，计算速度等方面仍由很多工作亟待解决，具体细节将在工作展望部分进行介绍。

6.2 工作展望

在下一步的工作中，将针对该模拟仿真系统的如下几个方面进行优化与改进：

（1）系统的多样性：

目前本系统的垃圾实体与环卫工实体均具有相同的特征参数。而在现实情况中，考虑到不同的垃圾有不同的处理方式以及不同的处理时间，而且每个环卫工的工作能力也不尽相同。在下一步的开发中，系统将会讲这种多态性的特征纳入，以达到更真实的模拟效果。

（2）系统的可靠性：

对于运行时的异常检测与处理，本系统目前还没有设计相关的机制来进行控制。虽然目前来说，在数据量较小的情况下，系统运行正常，但考虑到今后该仿真系统规模与功能的不断复杂化，提高系统的可靠性是十分必要的。

在下一步的设计中，将设计相应的异常除处理模块并对相应各全局链表数据结构进行封装，以提高整个系统的安全性。

（3）系统的资源利用率：

目前来说，本系统在资源利用率上仍有许多浪费，比如各链表之间是通过拷贝复制实现实体的转移，导致在空间复杂度上有较大的浪费。在下一步中，可以考虑使用指针或引用技术实现各链表之间实体的转移，这样会大幅提升系统的资源利用率。

（4）系统的可扩展性：

考虑到现实世界中，垃圾的产生与报告（行人发现，无人机拍摄等）是属于系统外生流程，可以通过提供相应的接口实现外部信息的导入。这需要进一步提高各模块之间接口的通用性，使得系统可以按照不同的功能需求进行相应的扩展改造。

（5）系统的 UI 界面开发：

由于开发时间有限，本系统只是满足了基本的功能要求，实现了相关的内核模块。在下一步的开发中，能否设计一个能够进行实时交互的 UI 界面将是未来工作的一个主要方向。

致　谢

短短的两个多月，从最开始的方向确定到最后毕业论文的完成。虽然时间有些仓促，但在整个过程中我也有很多的收获。

一直以来我对计算机方面有着很浓厚的兴趣，本科阶段也因为这个原因修读了计算机科学与技术二学位。考虑到我本身的主专业是物流管理，所以在选题方面在导师的建议下选择了一个有着较强融合性的方向。

在整个课程设计阶段，我要感谢我的导师姚杰老师。虽然我平时有写过一些小的程序，但从没有开发过多线程的模拟仿真框架，对于面向对象的开发模式我也是第一次接触。在设计框架的时候，姚杰老师给我提供了不少的帮助与支持，让我对整个框架的设计思路有了更清晰的认识。而在开发的过程中，在我遇到困难的时候，姚老师也能耐心地给我提供相应的指导。在这里，我向姚杰老师表示我最真挚的谢意。

此外，我还要感谢计算机学院负责双学位的各位领导与老师能给我提供这项的学习机会。在这两年的时间中，虽然很辛苦，但我也学到了很多的知识与技能，让我对整个计算机领域方向有了更深更全面的认识，在看待问题时也有了新的角度。我相信在今后的学习和工作中，这也将会是一笔宝贵的财富。

参考文献

[1] 范文慧，吴佳惠. 计算机仿真发展现状及未来的量子计算机仿真. 系统仿真学报. 第 29 卷第 6 期

[2] 董军,胡上序,陈德钊. 面向对象的计算机仿真建模. 计算机工程与应用. 1997 年 3 月.

[3] 邓书晶. 计算机离散事件仿真的原理. 计算机与现代化. 2009 年第 6 期.

[4] 赵阳. 计算机支持的协同工作中的并发控制研究[硕士学位论文]. 山东:山东师范大学,2006.

[5] 房毓菲,单志广. 智慧城市顶层设计方法研究及启示. 电子政务 EGOVERNMENT 2017 年第 2 期（总第 170 期）

[6] 葛蕾蕾 佟姮 侯为刚. 国内智慧城市建设的现状及发展策略. 行政管理改革 2017.7

[7] 王念社. 大型餐厅服务系统离散仿真研究[硕士学位论文]. 黑龙江:哈尔滨工程大学,2008

[8] Kristin L.Sainani PhD. What is Computer Simulation? PM&R Volume 7, Issue 12, December 2015, Pages 1290-1293.

[9] Brent H. Kinghorn. Computer Simulation: What’s the Story? American Journal of Management vol. 13(2) 2013.

[10] J. RICHARD HARRISON, ZHIANG LIN, GLENN R. CARROLL, KATHLEEN M. CARLEY. SIMULATION MODELING IN ORGANIZATIONAL AND MANAGEMENT RESEARCH. Academy of Management Review 2007, Vol. 32, No. 4, 1229-1245.

[11] 庞丽萍，阳富民. 计算机操作系统（第二版）. 人民邮电出版社. 2014.1

[12] Stephen Prata, 张海龙，袁国忠译. C++ Primer Plus（第6版）. 人民邮电出版社. 2012.7

[13] 胡斌，周明. 管理系统模拟. 清华大学出版社. 2008.7