ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение высшего профессионального образования

«ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Б. Г. Ослин

Моделирование

Имитационное моделирование СМО

Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета

Издательство
Томского политехнического университета
2010

УДК 519.8(075.8)
ББК 22.18я73
О-72

Ослин Б.Г.

О-72 Моделирование. Имитационное моделирование СМО: учебное пособие / Б. Г. Ослин; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2010. – 128 с.

В учебном пособии рассматривается технология создания программ имитационного моделирования систем массового обслуживания на универсальных языках программирования. Данная технология может быть использована для моделирования любых систем, функционирование которых описывается дискретно-непрерывными процессами. В пособии также приведены основные сведения из теории массового обслуживания и рассмотрены методы и алгоритмы моделирования случайных величин с произвольными и конкретными законами распределения.

Предназначено для студентов специальности «Информационные системы и технологии».

УДК 519.8(075.8)
ББК 22.18я73

Рецензенты

Доктор физико-математических наук, профессор ТГУ
В. И. Биматов

Кандидат технических наук, доцент ТПУ
Ю. Я. Кацман

Введение

Моделирование есть исследование объектов познания на их моделях, т. е. построение и изучение моделей как реально существующих предметов и явлений, так и конструируемых объектов и систем. Другими словами, это замещение одного объекта (оригинала) другим объектом (моделью) с целью получения информации об определенных свойствах оригинала.

Модель (в широком смысле) есть образ или прообраз объекта, системы или явления, используемый при определенных условиях в качестве их заменителя или представителя. Это представление объекта, системы или явления в некоторой форме, отличной от формы их реального существования. При этом под термином «реальный» следует понимать существующий или способный принять ту или иную форму существования. Так, моделью Земли служит глобус, моделью полета ракеты является система дифференциальных уравнений, описывающих движение ракеты по траектории, чучело животного есть модель этого животного, а фотография в паспорте и другие паспортные и анкетные данные – модель владельца паспорта. С другой стороны, живописец, например, называет моделью именно изображаемого им человека.

Можно привести и другие определения понятия «модель», например, такое: объект M является моделью объекта A относительно некоторой системы S характеристик (свойств), если M строится (или выбирается) для имитации A по этим характеристикам.

Как следует из приведенного определения, моделирование – это метод познания и, следовательно, целенаправленный процесс. По существу, всякое познание есть моделирование. Человек получает через органы чувств информацию из внешней среды, о происходящих в ней явлениях, осмысливает ее, в результате в сознании появляются образы (модели) этих явлений.

Как метод научного исследования моделирование зарождается в античную эпоху одновременно с возникновением научного познания. В отчетливой форме (хотя без употребления самого термина) моделирование начинает широко использоваться в эпоху Возрождения архитекторами, скульпторами, в научных исследованиях. Со времен Ньютона метод моделирования используется уже вполне осознанно, а в XIX – XX веках трудно назвать область науки или ее приложений, где моделирование не имело бы существенного значения. Особенно это относится к физике, химии и другим точным наукам, которые стали классическими «полигонами» методов моделирования. Появление же

первых электронных вычислительных машин и формулирование основных принципов кибернетики привело к поистине универсальной значимости новых методов моделирования – как в абстрактных областях знаний, так и в их приложениях. Моделирование ныне приобрело общенаучный характер и применяется в исследованиях живой и неживой природы, в науке о человеке и обществе. Особое значение моделирование приобретает в сфере управления различными объектами, где основными являются процессы принятия решений на основе получаемой информации.

Моделирование необходимо предполагает использование абстрагирования и идеализации. Отражая существенные (с точки зрения цели исследования) свойства оригинала и отвлекаясь от несущественного, модель выступает как некоторый абстрактный идеализированный объект. При этом от характера и уровней, лежащих в основе моделирования абстракций и идеализаций, в большей степени зависит весь процесс переноса знаний с модели на оригинал. Но каков бы ни был уровень абстракции при моделировании следует считаться с тем, что моделирование оригинала никогда не даст полного знания о нем.

Моделирование всегда используется вместе с другими общенаучными и специальными методами. Прежде всего, моделирование тесно связано с экспериментом. Изучение оригинала на его модели можно рассматривать как особый вид эксперимента: «модельный эксперимент». Отличие модельного эксперимента от «прямого» (обычного эксперимента на оригинале) заключается еще и в том, что модель является одновременно и средством, и объектом экспериментального исследования, заменяющим изучаемый объект. Модельный эксперимент позволяет изучать такие объекты, прямой эксперимент над которыми затруднен, экономически невыгоден, либо вообще невозможен в силу тех или иных причин.

Единая классификация видов моделирования затруднительна в силу многозначности понятия «модель». Ее можно проводить по различным основаниям: по характеру моделей (т. е. по средствам моделирования); по характеру моделируемых объектов; по сферам приложения (моделирование в технике, в физических науках, в химии, моделирование психики и т. д.), по уровням (глубине) абстракции и т. д. В связи с этим любая классификация методов моделирования обречена на неполноту, тем более, что терминология в этой области опирается не столько на строгие правила, сколько на научные и практические традиции, а еще чаще определяется в рамках конкретного контекста и

вне его никакого стандартного значения не имеет. Тем не менее, рассмотрим некоторые классификации.

В основе моделирования лежит *теория подобия,* которая утверждает, что абсолютное подобие может иметь место лишь при замене одного объекта другим точно таким же. При моделировании абсолютное подобие используется достаточно редко. Обычно стремятся к тому, чтобы модель достаточно хорошо в соответствии с поставленной целью отображала исследуемую сторону объекта. Поэтому в качестве одного из первых признаков классификации видов моделирования можно выбрать степень полноты или уровень *абстракции и идеализации* модели. Разделять модели в соответствии с этим признаком на какие-либо отдельные группы (например, полные модели, неполные, приближенные) вряд ли целесообразно. Существуют или имеют право существовать модели всех возможных уровней абстракции, от самых абстрактных, самых примитивных, до абсолютных копий оригинала.

По характеру той стороны объекта, которая подвергается моделированию, уместно различать *моделирование структуры* объекта и *моделирование поведения.* Это различие сугубо относительно для химии или физики, но оно приобретает четкий смысл в науках о жизни и в кибернетике. Например, при «кибернетическом» моделировании обычно абстрагируются от структуры, рассматривая ее как «черный ящик», модель которого строится в терминах соотношения между состояниями его «входов» и «выходов». Входы соответствуют внешним воздействиям на изучаемую систему, выходы – ее реакциям на них, т. е. поведению.

По форме представления оригинала модели можно поделить на предметные и знаковые. *Предметным* называется моделирование, в ходе которого исследование ведется на модели, воспроизводящей основные геометрические, физические, динамические и функциональные характеристики оригинала и выполненные в виде реальных изделий. К таким моделям относятся всевозможные уменьшенные или увеличенные копии оригиналов, выполненные обычно не из тех материалов, из которых сделаны оригиналы. Это то, что в обиходе собственно и называется моделью. Например, при создании новых самолетов создаются их уменьшенные модели из металла или другого материала, повторяющие все его геометрические формы и пропорции. Затем на этих моделях проводят эксперименты в аэродинамических трубах и таким образом проверяют и оценивают возможности конструкции будущего самолета. На завершающих

стадиях создания нового самолета всегда создается почти полная копия самолета, на которой проводят летные испытания.

К предметным моделям относятся также всевозможные макеты зданий и сооружений, которые вместе с чертежами и прочей документацией используются при строительстве. Всевозможные тренажеры, применяемые для обучения и тренировки летчиков, моряков, водителей локомотивов и т. д., также можно с полным правом считать предметными моделями.

При *знаковом* моделировании моделями служат знаковые обозначения какого-либо вида: схемы, графики, чертежи, формулы, слова и предложения на некотором естественном или искусственном языке и т. д. Знаковые обозначения и их элементы всегда рассматриваются вместе с определенными преобразованиями, операциями над ними (преобразования математических, логических, химических формул, выполнение компьютерных программ, создание трехмерных изображений архитектурных сооружений по имеющимся чертежам отдельных элементов и т. д.). С другой стороны, действия со знаками в той или иной мере связаны с пониманием знаковых обозначений и их преобразований. Формулы, математические уравнения и другие выражения, применяемого при построении модели научного языка, определенным образом интерпретируются (истолковываются) в той предметной области, к которой относится оригинал.

Одной из разновидностей знакового моделирования является мысленное или интуитивное моделирование, использующее мысленно-наглядное представление знаков и операций над ними или не использующее никаких четко фиксированных знаковых систем, а протекающее на уровне «модельных представлений». Такое моделирование есть непременное условие любого познавательного процесса на его начальной стадии.

Важнейшим видом знакового моделирования является *математическое* моделирование, осуществляемое средствами языка математики и логики. Под математическим моделированием будем понимать процесс установления соответствия реальному объекту некоторого математического объекта, называемого математической моделью, и применение соответствующих математических методов исследования этой модели с целью получения необходимых данных о реальном объекте.

В данном учебном пособии рассматривается достаточно узкий класс математических моделей, которые называются системами массового обслуживания (СМО) и применение метода имитационного моделирования для исследования процессов функционирования таких систем. В первой главе пособия приведены основные сведения из теории массового обслуживания, определены основные понятия, перечислены основные элементы, которые должны быть определены при создании любой СМО, дана краткая классификация СМО и методов их исследования.

Следующая глава пособия посвящена непосредственно имитационному моделированию. В ней подробно рассматривается предложенная автором технология создания имитационных моделей в виде компьютерных программ на универсальных языках программирования (С++, Паскаль и др.). Рассматриваются этапы создания имитационной модели, универсальная структура программы моделирования.

В следующей главе приведены основные методы имитации случайных чисел и алгоритмы, используемые в компьютерных программах для моделирования и преобразования случайных величин с различными законами распределения.

И, наконец, в последней главе рассмотрены примеры создания систем массового обслуживания для нескольких реальных систем и тексты на языке «Си» основных модулей программ имитационного моделирования этих систем.

Глава 1

Основы теории массового обслуживания

Основные понятия и определения

Системы массового обслуживания (СМО) – это математические модели таких реальных или проектируемых систем, функционирование которых можно рассматривать как последовательное взаимодействие неких дискретных объектов с элементами системы. Эти объекты, обычно поступающие в систему из внешних источников, некоторое время взаимодействуют с элементами системы по определенным правилам, а затем покидают систему.

В СМО дискретные объекты называются заявками, требованиями или клиентами. Их взаимодействие с элементами системы в течение времени, заданного некоторым образом, называется обслуживанием, а сами эти элементы – узлами обслуживания или обслуживающими приборами. Порядок взаимодействия заявок с системой, т. е. порядок их поступления в систему, пребывания в системе, порядок обслуживания и ухода из системы, правила работы обслуживающих приборов и т. п., называется дисциплиной обслуживания.

Возьмем в качестве примера системы, моделью которой может быть СМО, обычный магазин с торговым залом, витринами и несколькими отделами. Покупатели приходят в магазин, какое-то время рассматривают витрины, что-то покупают в одном или нескольких отделах и уходят. Дискретные объекты в этой системе – это покупатели. Элементы системы, с которыми они взаимодействуют некоторое время, – это продавцы в торговых отделах, витрины и образцы товаров в торговом зале, которые посетители разглядывают некоторое время прежде, чем стать покупателями или уйти из магазина без покупок. Правила взаимодействия объектов с элементами системы есть порядок обслуживания покупателей.

Дискретными объектами в этой системе могут быть не только покупатели, но и товары, которые поступают в магазин из некоторых внешних источников, какое-то время ожидают своего покупателя, затем взаимодействуют с продавцами и покидают магазин вместе с покупателями.

Другой пример – поликлиника. Дискретные объекты – пациенты. Элементы, с которыми они взаимодействуют, – регистратура, врачи, лаборатория, процедурные кабинеты и т. д. В каждой поликлинике свой

порядок обслуживания пациентов. В одной обслуживают только студентов, в другой только платное обслуживание по предварительной записи, в третьей обслуживают население определенного района в порядке живой очереди, в четвертой проводятся только осмотры с целью выяснения пригодности к той или иной профессии и т. д.

Вероятно, каждому ясно, что обслуживается, какими элементами системы и по каким правилам, когда речь идет, например:

а) о поступлении на ремонтную базу требующих технического ремонта транспортных средств, когда в зависимости от численности технического персонала и оборудования ремонтная база может одновременно обслуживать одно или несколько транспортных средств;

б) о прибытии самолетов в аэропорт, когда очень важно, чтобы самолеты прилетали и улетали строго по графику и не образовывались очереди на взлет и, тем более, на посадку;

в) о попытках дозвониться по телефону, когда набирающие телефонные номера индивидуумы нуждаются в обслуживании в виде телефонных разговоров;

г) о компьютерной сети, по которой передаются различные команды на выполнение программ на удаленных компьютерах, а результаты их выполнения в пакетном режиме передаются обратно.

Особенностью всех реальных или проектируемых систем, для исследование которых можно использовать модели типа СМО, является то, что процессы их функционирования представляют собой совокупность отдельных относительно кратковременных, «коротких» процессов. В каждый момент времени функционирования системы в ней могут происходить несколько таких коротких процессов. Эти процессы начинаются в различные моменты времени, длятся некоторое время, заканчиваются одни и начинаются другие. Каждый короткий процесс начинается в момент наступления некоторого события. Окончание каждого процесса также является событием, которое может быть началом другого короткого процесса.

Например, процесс функционирования магазина можно представить в виде совокупности следующих коротких процессов: ожидание прихода очередного покупателя, знакомство покупателя с представленными на витринах товарами, ожидание покупателем в очереди начала обслуживания, его обслуживание продавцом. При более детальном рассмотрении процесса функционирования можно выявить и другие короткие процессы или перечисленные выше разбить на еще более мелкие. Поскольку покупателей в магазине может быть много и

приходят они в разное время, то и коротких процессов в каждый момент времени также может быть много.

Целевое назначение СМО

Очевидно, что всякие реальные системы, в том числе и те, которые служат объектами для моделей типа СМО, имеют бесконечное множество свойств, параметров и характеристик. Известно также, что всякая модель создается и может использоваться для исследования только отдельных свойств и характеристик объекта моделирования. Следовательно, прежде чем создавать модель какой-либо системы, выбирать ее тип, уровень детализации, необходимо решить, для чего эта модель будет использоваться, какие проблемы будут решаться с ее помощью. Другими словами, сначала необходимо определить цель создания модели, и только после этого можно приступать к выбору типа модели, а затем к ее созданию и использованию для достижения поставленной цели.

Для каких целей можно использовать модели типа СМО, что можно исследовать на моделях этого типа? Во-первых, СМО создаются и используются с целью исследования процессов функционирования систем. Естественно, при этом моделируется и структура системы, но исследуются именно процессы функционирования системы с учетом ее структуры. Во-вторых, на моделях типа СМО исследуются только такие характеристики и свойства процессов функционирования систем, которые определяются длительностями отдельных происходящих в системе коротких процессов и моментами наступления в ней определенных событий. Другими словами, исследуются только те характеристики процесса функционирования объекта моделирования, которые зависят от времени.

На модели магазина можно исследовать среднюю длительность пребывания покупателя в магазине, степень занятости продавцов, длины очередей, вероятности ухода покупателей из магазина без покупок и другие зависящие от времени параметры процесса функционирования магазина. Такие исследования могут быть весьма полезны руководству магазина при выборе, например, наилучшей организации торговли.

Если необходимо исследовать пропускную способность компьютерной сети при различных уровнях загрузки, подобрать параметры передачи данных, оценить скорости передачи, то модель типа СМО может в этом помочь. В такой модели заявки будут представлять передаваемые по сети файлы, сообщения,

информационные и служебные пакеты и т. д. Обслуживающие приборы будут моделировать процессоры вычислительных устройств, каналы передачи данных, а дисциплина обслуживания – протоколы передач информации. Если же необходимо протестировать каналы связи на надежность и помехозащищенность или выявить ошибки в кодах программных модуле, то модели типа СМО вряд ли подойдут для этих целей.

Основные элементы СМО

К основным элементам любой СМО относятся:

- заявки;
- потоки заявок и различных воздействий на элементы системы;
- механизмы обслуживания;
- дисциплина обслуживания.

Заявки

Для задания СМО необходимо прежде всего определить, что будем считать заявками. Заявки в СМО должны моделировать все существенные признаки и параметры тех дискретных объектов, моделями которых они являются. Если для цели исследования безразличны какие-либо особенности этих объектов, то можно считать все заявки одинаковыми, однотипными. В общем случае заявки могут быть нескольких типов или отличаться друг от друга по одному и нескольким параметрам. От типа заявки или значений ее параметров может зависеть дисциплина и длительность обслуживания.

Например, в некоторых магазинах предусмотрен льготный порядок обслуживания отдельных категорий покупателей. Если необходимо учесть этот факт при моделировании, то надо определить, что в СМО обслуживаются заявки двух типов и дисциплины обслуживания их различны. Заявки одного типа обслуживаются по очереди, другого – сразу при поступлении или по отдельной, приоритетной очереди.

Если модель магазина предполагается использовать для исследования движения товаров, то поступающие в магазин виды товаров надо рассматривать как заявки определенных типов, отличных от типа заявок, которые моделируют покупателей. Каждая такая заявка-товар может иметь параметры, значения которых определяют, например, вес товара, объём, тип упаковки, срок хранения и т. д. Естественно, для этих заявок надо в СМО предусмотреть свои каналы поступления и, возможно, отдельные от других обслуживающие приборы.

Заявки в процессе обслуживания могут изменять свой тип и значения параметров, объединяться с другими заявками и образовывать новый тип, разбиваться на несколько заявок новых типов. Заявки могут не только поступать в СМО из некоторых внешних источников, но и генерироваться в самой системе по определенным правилам.

Потоки заявок и воздействий

При создании любой системы массового обслуживания, в которую заявки поступают из внешних источников, необходимо задать законы, определяющие порядок поступления заявок, их типы и параметры, т. е. задать потоки заявок в систему. Проще всего поток заявок можно задать в виде строгого графика, который точно определяет моменты поступления заявок, их количество в каждый момент, их типы и параметры. Например, при моделировании аэропорта заявками можно считать самолеты, которые должны прибывать в аэропорт по расписанию. Вполне естественно предположить, что это расписание и есть тот закон, который определяет поток заявок в СМО, моделирующую аэропорт.

Чаще всего приходится предполагать, что приход заявок в СМО зависит от внешних непредсказуемых обстоятельств. Лучшее, что можно сделать в этом случае, так это рассматривать факты появления заявок как случайные события во времени, т. е. представить их в виде одного или нескольких потоков случайных событий.

Потоком событий называется последовательность событий, наступающих одно за другим в моменты времени t_0, t_1, t_2, …, t_k, … . В общем случае длительности интервалов времени между очередными событиями $\tau_i = t_i - t_{i-1}$, $i = 1, 2, \ldots$ считаются случайными, и поэтому такие потоки называют потоками случайных событий. Детерминированный, или регулярный, поток событий, когда события наступают через вполне определенные, фиксированные интервалы времени, можно рассматривать как частный случай потока случайных событий.

Для того чтобы задать поток событий, т. е. определить его модель, необходимо задать законы распределения F_i случайных длительностей интервалов времени между очередными событиями потока. Эти распределения и их свойства полностью определяют особенности и тип потока.

Поток случайных событий называется *рекуррентным*, если все законы F_i одинаковые, т. е. $F_i = F$ для всех i. Другими словами, рекуррентный поток представляет собой такую последовательность

событий, для которой все интервалы между событиями «ведут себя» как бы одинаково, т. е. подчиняются одному и тому же закону распределения. Термин «рекуррентность» означает в данном случае повторяемость (в статистическом смысле) свойств интервалов времени между очередными событиями.

Для потоков случайных событий, кроме вероятностных характеристик интервалов между заявками, вводят в рассмотрение вероятность P_n наступления некоторого числа n событий в заданном интервале времени. *Поток без последействия* характеризуется тем, что для двух, не пересекающихся интервалов времени $(t_1, t_1 + \Delta t_1)$ и $(t_2, t_2 + \Delta t_2)$, где $\Delta t_1 > 0$, $\Delta t_2 > 0$ и $t_2 > t_1 + \Delta t_1$, вероятность наступления n событий на втором интервале не зависит от числа наступивших событий на первом интервале. Отсутствие последействия означает отсутствие вероятностной зависимости между будущим течением процесса наступления событий и его прошлым.

Например, приход покупателей в магазин вполне можно считать потоком без последействия, если покупатели идут из разных, независимых мест, а не с только что закончившегося сеанса в соседнем кинотеатре. Появления троллейбусов на остановке, очевидно, нельзя считать потоком без последействия. Если троллейбусов нет какое-то время, то скорее всего потом они пойдут один за другим.

Потоком *с ограниченным последействием* или потоком *Пальма* называется такой поток, в котором интервалы между событиями независимы. Другими словами, это потоки, в которых вероятность наступления события на интервале времени Δt зависит от величины этого интервала и момента наступления предыдущего события, но не зависит от моментов наступления более ранних событий.

Поток называется *стационарным*, если вероятность появления какого-либо числа событий на интервале времени $(t, t + \Delta t)$ зависит только от длины этого интервала Δt и не зависит от t, т. е. от расположения интервала на оси времени. Для стационарного потока среднее число событий, наступающих в единицу времени, постоянно.

Не надо путать отсутствие последействия и стационарность. Например, число соединений, выполняемых телефонной станцией с 13 до 14 часов значительно больше, чем с 1 до 2 часов ночи. Поток выполняемых соединений явно не стационарный. Но количество соединений с 13 до 14 часов не зависит (или можно считать, что не зависит) от количества соединений с 0 до 1 часа. Возможно и наоборот, поток может быть стационарным и обладать последействием. Пример

тому регулярный поток, когда события наступают через интервалы времени постоянной длины.

Если заявки в потоке однотипные, без параметров и поступают по одной, то задание потока событий полностью определяет поток заявок. В общем случае необходимо задать законы, определяющие количество, типы и параметры поступающих заявок при каждом событии потока.

Поток заявок называется *ординарным*, если все заявки в потоке поступают по одной. Если поток неординарный, то необходимо задать закон, который бы определял количество заявок, поступающих при каждом событии потока. Количество поступающих одновременно заявок можно задавать постоянным, зависящим от состояния системы и ее параметров, случайным, т. е. определяться некоторым дискретным распределением вероятностей.

Поток заявок называется *однородным*, если заявки в потоке одного типа. В неоднородном потоке заявки могут быть нескольких типов или обладать разными свойствами. Например, моделью входного потока покупателей в магазин может быть поток однородных событий, если в системе массового обслуживания, моделирующей работу магазина, не учитываются индивидуальные особенности покупателей. Если же необходимо учитывать особенности обслуживания некоторых категорий покупателей, то в качестве модели входного потока следует выбрать неоднородный поток событий. В нем одни события являются фактами прихода обычных покупателей, а другие – покупателей, которые обслуживаются особым образом.

Для задания неоднородного потока, кроме закона распределения случайных длительностей интервалов времени между очередными неоднородными заявками, необходимо задать закон, определяющий типы и параметры заявок. Очевидно, что неоднородный поток, например, двух типов заявок можно представить и как суммарный поток, образованный наложением двух однородных потоков. Но задание двух однородных потоков не определяет неоднородный поток, как результат их наложения, если по законам распределения длительностей интервалов времени между событиями в однородных потоках нельзя однозначно определить закон распределения длительностей интервалов времени между очередными неоднородными событиями в суммарном потоке.

Особое место в теории массового обслуживания занимает так называемый *простейший или пуассоновский поток* заявок. Такое положение объясняется тем, что существует теорема, согласно которой сумма большого числа независимых потоков с произвольными законами

распределения длительностей интервалов между событиями приближается к простейшему.

Простейшим потоком называется поток, обладающий тремя свойствами: ординарностью, стационарностью и отсутствием последействия. Эти свойства позволяют утверждать, что функция распределения длины промежутка между двумя последовательными наступлениями событий потока равна:

$$F(t) = 1 - e^{-\lambda t},$$

т. е. длины интервалов между очередными событиям потока имеют экспоненциальное распределение, а вероятность наступления ровно k событий на интервале времени t равна

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t},$$

где λ – некоторое положительное число, которое носит название *параметра потока*.

Легко подсчитать, что для простейшего потока среднее число событий, наступающих за время t, равно:

$$\sum_{k=1}^{\infty} kP_k(t) = e^{-\lambda t} \sum_{k=1}^{\infty} k \frac{(\lambda t)^k}{k!} = e^{-\lambda t} \lambda t \sum_{k=1}^{\infty} \frac{(\lambda t)^{k-1}}{(k-1)!} = \lambda t \, e^{-\lambda t} \, e^{\lambda t} = \lambda t.$$

Математическое ожидание числа событий, наступающих в единицу времени, называется *интенсивностью потока*. Из последнего соотношения видно, что для простейшего потока интенсивность совпадает с параметром потока. Очевидно также, что средняя длительность интервала времени между очередными событиями простейшего потока равна $1/\lambda$.

Выбор законов F_i, определяющих длительности интервалов времени между очередными событиями потока, т. е. собственно определяющие поток событий, в общем случае является достаточно сложной задачей. Необходимо учитывать особенности той реальной или проектируемой системы, для моделирования которой создается система массового обслуживания. Но в некоторых случаях это можно сделать просто.

Предположим, например, что необходимо задать поток заявок в систему массового обслуживания, моделирующую работу некоторого магазина с 11 до 12 часов дня. Пусть заявки в этой СМО моделируют покупателей. Все заявки однотипные, без параметров, поступают по

одной и одним потоком. Естественно предположить, что заявки поступают не строго по графику, а случайно. Естественно также предположить, приход очередной заявки совершенно не зависит от того, когда пришла предыдущая заявка. Пусть нам каким-то образом известно, что в среднем в единицу времени, например, одну минуту, в магазин приходит m покупателей, т. е. интенсивность прихода покупателей равна m, а средняя длительность интервала времени между приходом очередных покупателей соответственно равна $1/m$. В этом случае мы имеем все основания предполагать, что поток заявок в СМО пуассоновский интенсивности m.

Вернемся к примеру с аэропортом. Самолеты в аэропорт должны прибывать строго по расписанию и поэтому предположение о строгом графике поступления заявок-самолетов в СМО, моделирующую аэропорт, вполне естественно. Но фактически самолеты в зависимости от погоды и других обстоятельств немного опаздывают или прилетают чуть-чуть раньше. Если при моделировании необходимо учесть этот факт, то можно предположить, что длительность каждого интервала времени между приходами очередных заявок является случайной величиной, распределенной по нормальному закону с математическим ожиданием, равным длительности интервала по графику и некоторой небольшой дисперсией.

При создании СМО реальной системы, если есть возможность, можно провести статистическое исследование поступления дискретных объектов в систему и на основании полученных данных построить, например, гистограмму распределения длительности интервалов между моментами поступления очередных объектов, то эту гистограмму можно принять в качестве закона, определяющего поток заявок.

Если исследование создаваемой СМО предполагается с помощью имитационного моделирования, то иногда удобнее полагать, что длительности интервалов между очередными событиями потока, количества, типы и параметры поступающих заявок определяются не законами, а некоторыми алгоритмами. Это могут быть алгоритмы моделирования значений случайной величины, распределенных по заданным законам, или любые другие алгоритмы. Такие алгоритмы реализуются в программах моделирования в виде соответствующих процедур. При каждом обращении к такой процедуре она возвращает программе соответствующее значение. Это либо длительность интервала времени между очередными событиями потока, либо количество одновременно поступающих заявок, либо тип и параметры очередной заявки.

Кроме входных потоков заявок, часто для задания СМО, необходимо определять потоки различных событий, влияющих на работу системы. Такие события будем называть внешними воздействиями. К внешним воздействиям или событиям относятся, например, выходы из строя и восстановление обслуживающих приборов, переключения режимов их работы, появление дополнительных обслуживающих приборов и другие изменения в системе, наступающие через интервалы времени, определенные некоторым законом. Каждое внешнее воздействие в общем случае можно рассматривать как заявку определенного типа. Как и входные потоки заявок, эти потоки однозначно задаются законами, определяющими длительности интервалов между очередными событиями, или алгоритмами, позволяющими вычислять эти длительности в программах моделирования, а также характеристиками самих этих событий.

Механизм обслуживания

Системы массового обслуживания различаются количеством узлов обслуживания и числом обслуживающих приборов в них, количеством одновременно обслуживаемых требований, продолжительностью и типом обслуживания. Узлы обслуживания в СМО могут иметь самую разнообразную структуру. Под узлом обслуживания понимают обычно один или несколько обслуживающих приборов и комплекс различных элементов СМО (прежде всего очередей), которые регулируют порядок обслуживания поступающих в узел заявок.

Под обслуживающим прибором понимают то, что непосредственно взаимодействует с заявками некоторое время. Суть этого взаимодействия или обслуживания обычно не имеет значения. Иногда важен результат обслуживания, который выражается в изменении типов и параметров обслуженных заявок, и всегда важна длительность взаимодействия, которая должна быть определена. Именно такое взаимодействие или такой процесс взаимодействия заявки с элементом СМО, длительность которого определена некоторым законом, называется обслуживанием, а элемент, с которым взаимодействует заявка, называется обслуживающим прибором. Если при создании СМО было принято решение учитывать некоторый «короткий» процесс как часть процесса функционирования моделируемой реальной системы и для этого «короткого» процесса выбран закон, определяющий его длительность, то в СМО должен быть обслуживающий прибор, в котором длительность обслуживания определяется этим законом.

В теории массового обслуживания, как правило, предполагается, что длительности обслуживаний в приборах есть случайные величины. В этом случае для прибора в создаваемой СМО необходимо задать законы распределения случайных длительностей обслуживания заявок в этом приборе. Если предполагать, что длительности обслуживаний в приборе не случайные, то длительности обслуживаний в нем можно задавать постоянными, определяемыми некоторым графиком, однозначно зависимыми от типов и параметров заявок и т. д.

При имитационном исследовании СМО, как и в случае с потоками событий, проще и удобнее длительности обслуживания заявок в приборах определять алгоритмами их вычисления, которые реализуются в программах моделирования соответствующими процедурами или функциями.

Например, моделировать обувной отдел универмага можно двумя приборами. Один прибор моделирует обслуживание покупателей в торговом зале, когда они только выбирают и примеряют обувь, другой – кассу, где покупатели оплачивают покупку. Первый прибор может одновременно обслуживать до N заявок. (Для этого надо предположить, что больше N покупателей в отдел не допускают.) Следовательно, чтобы задать первый прибор, необходимо задать конкретное значение N (или закон, позволяющий вычислить это значение), и задать, например, закон распределения случайной длительности обслуживания (пребывания) каждой заявки в приборе.

Относительно второго прибора, естественно, предположить, что он в каждый момент времени может обслуживать только одну заявку и длительности обслуживаний всех заявок, случайные величины, распределенные, например, по нормальному закону с заданными параметрами.

Моделировать обслуживание покупателей в торговом зале можно не одним прибором, обслуживающим до N заявок одновременно, а узлом обслуживания, состоящим из N параллельно работающих однотипных приборов, каждый из которых моделирует место, занимаемое покупателем в торговом зале. В каждый момент времени любой из этих N приборов обслуживает только одну заявку (одного покупателя), либо находится в состоянии ожидания прихода заявки, либо уже обслужил заявку, но не освободился, поскольку заявка не покинула систему. Покупатель не освободил место в торговом зале, а находится в очереди ко второму прибору или уже обслуживается в нем. Длительности обслуживаний всех заявок во всех приборах являются случайными величинами с одним и тем же законом распределения.

Еще один пример. Торговая палатка, в которой один продавец и очередь с нетерпеливыми покупателями. Покупатели подходят к палатке в случайные моменты времени, становятся в очередь. Постаяв в очереди некоторое время, покупатель может уйти, так ничего и не купив. При моделировании работы палатки следует предположить, что заявки моделируют покупателей, а обслуживающий прибор – продавца. А вот очередь покупателей можно моделировать по-разному. Можно предположить, что заявки образуют обычную очередь к прибору, но на нее воздействует некоторый поток случайных событий, и каждое событие из этого потока вызывает уход из очереди с равной вероятностью любой заявки. Можно также предположить, что на каждую заявку в очереди воздействует отдельный поток внешних событий, состоящий всего из одного события. Это событие наступает через случайное время после поступления заявки в очередь. Если заявка успевает поступить на обслуживание в прибор до наступления этого события, то поток исчезает. Если событие наступает до начала обслуживания, то поток тоже исчезает, а вместе с ним и заявка покидает очередь.

Другой вариант моделирования очереди с нетерпеливыми заявками может быть такой. В состав СМО кроме прибора, моделирующего продавца, вводят еще N приборов, которые можно интерпретировать как таймеры. Для каждой поступающей в очередь заявки включается один из этих приборов-таймеров на некоторое случайное время, определяемое по некоторому заданному закону. Работу таймера в этом случае следует рассматривать как специфическое обслуживание или как ожидание некоторого события. Если таймер срабатывает, то соответствующая заявка уходит из очереди, а таймер отключается. Если заявка до срабатывания таймера успевает поступить на обслуживание в прибор, то таймер также отключается и впоследствии может быть запущен для любой другой поступившей в систему заявки.

Обслуживающие приборы в СМО могут моделировать не только взаимодействие неких дискретных объектов-заявок с элементами реальной системы, но и генерацию заявок в самой системе, а также ожидание некоторых событий в системе. Например, приготовление блюд в кафе в предположении, что исходных продуктов предостаточно, можно моделировать одним или несколькими приборами, на вход которых не поступают никакие заявки. По окончании обслуживания в каждом из таких приборов в СМО поступают партии заявок, которые моделируют приготовленные блюда.

Ожидание клиентами некой ремонтной мастерской исполнения их заказов можно моделировать системой массового обслуживания, в которой кроме прочих обслуживающих приборов, моделирующих прием заказов от клиентов, их исполнение работниками мастерской, выдачу заказов клиентам и т. д., есть еще N однотипных приборов, моделирующих ожидание клиентами исполнения их заказов. Количество приборов N равно максимальному количеству одновременно выполняемых заказов. Пока заказов-заявок нет, все эти приборы свободны. Как только заказ принят к исполнению, один из свободных приборов запускается в работу на время исполнения заказа или на тот период времени, когда ожидается приход клиента за выполненным заказом.

При создании СМО часто приходится учитывать тот факт, что один и тот же элемент некой реальной системы выполняет различные функции. В этом случае в качестве модели такого элемента в СМО можно рассматривать прибор, в котором выполняются различные типы обслуживаний или обслуживание заявки в нем выполняется в несколько этапов. Обслуживание может прерываться на некоторое время, в течение которого прибор не обслуживает заявку, но и не может взять другую на обслуживание. Тип обслуживания может зависеть от типа или параметров заявок, от канала ее поступления, от того, какие этапы обслуживания прошла заявка, и т. д. Длительности обслуживаний различных типов могут определяться как одним законом или алгоритмом, так и совершенно различными. Возможно, что и результаты обслуживаний различных типов, могут приводить к изменениям в поведении заявок или системы в целом.

В заключении отметим, что обслуживанием в СМО называется любой процесс, длительность которого в целом или каждого из его этапов определяется некоторым законом или алгоритмом. Эти законы или алгоритмы, их параметры должны быть заданы при создании СМО. Любое облуживание выполняется в каком-либо обслуживающем приборе. Если исследование СМО предполагается методами имитационного моделирования, то целесообразно предполагать, что в каждом обслуживающем приборе в каждый момент времени может находиться только одна заявка.

Дисциплина обслуживания

Обычно дисциплина задается в виде словесного описания всех без исключения правил, законов и алгоритмов поведения заявок и взаимодействия всех элементов СМО. Для пояснения, если необходимо,

рисуется схема, на которой изображаются каналы поступления заявок, обслуживающие приборы, накопители для заявок или очереди, каналы продвижения заявок и т. д. Дисциплина обслуживания – это полное описание правил функционирования СМО. При этом в дисциплине обслуживания нет необходимости уточнять тип, характер или природу обслуживания заявок. Важно только, когда и как поступают заявки в систему, как организован их прием, как и где они ожидают начала обслуживания, что происходит с ними после обслуживания и т. д. Другими словами, важно только, какие события происходят в системе и как они взаимосвязаны во времени.

Поскольку в теории массового обслуживания важнейшим понятием является очередь, то дисциплина обслуживания – это, прежде всего, описание очередей, т. е. описание правил приёма заявок и постановок их в очереди, поведения заявок в очереди и правила выбора заявок из очередей на обслуживание и т. д.

Приём заявок в СМО может быть двух типов: либо на обслуживание принимаются все поступающие заявки, либо для каждой заявки проверяется условие ее приема, т. е. СМО работает с отказами в обслуживании. Условия приема заявок могут быть самыми различными. Заявка может быть не принята, если общее количество находящихся в системе заявок равно некоторому значению N, или до этого в систему уже поступили N заявок, или ограничен каким-либо образом накопитель для поступающих заявок и т. д. Заявка может быть принята в систему с некоторой вероятностью, которая может зависеть от количества заявок в системе или других величин. Ограничение на прием заявок могут определяться не только количеством заявок, но суммарным значением некоторого параметра всех заявок, находящихся в накопителе. Например, если заявки являются моделями файлов и для каждой заявки определен параметр, моделирующий длину файла в байтах, то условие приема заявок может определяться размером накопителей для файлов, измеряемых в байтах.

В реальных системах обслуживания накопители для заявок и очереди всегда ограничены. В СМО иногда удобно пренебречь ограничениями, особенно в тех случаях, когда проводится аналитическое исследование. При имитационном моделировании наоборот чаще всего рассматриваются СМО с отказами, поскольку бывает очень сложно пренебречь размерами накопителей.

Принятая в СМО заявка либо сразу поступает на обслуживание в соответствующий прибор, либо становится в одну из очередей. Если заявки из потока или после обслуживания в некотором приборе

направляются в разные очереди, то в дисциплине обслуживания должно быть указано правило постановки заявки в ту или иную очередь. Например, поступающая заявка может быть направлена в более короткую очередь, а если очереди равны, то с равной вероятностью в любую из них. В общем случае правила постановки заявок в очереди могут быть какими угодно, но в любом случае эти правила должны однозначно определять этот порядок.

Правила организации очередей в СМО также могут быть какими угодно, но основных правила три: «первым пришел – первым обслуживаешься», «последним пришел – первым обслуживаешься» и «случайный отбор заявок». Первое правило моделирует знакомые всем нам очереди, когда вновь поступающие заявки становятся в конец очереди, а выбор заявки на обслуживание производится из начала очереди. Второе правило означает, что поступающая заявка становится первой в очереди, отодвигая ранее поступившие, и выбор на обслуживание также производится из начала очереди. Третье правило моделирует накопитель, в котором поступающие заявки собираются в некий накопитель, а из него случайным образом выбираются на обслуживание. При необходимости дисциплиной обслуживания могут быть определены правила переходов заявок из очереди в очередь, уходы из очереди нетерпеливых заявок, переходы заявок с одного места на другое и т. д.

Выбор заявки на обслуживание прибором, перед которым только одна очередь, очень простой. Закончив обслуживание одной заявки прибор мгновенно берет на обслуживание следующую. Только всегда следует оговаривать в дисциплине обслуживания, где находится обслуживаемая заявка: в очереди или в приборе. Например, в магазине покупатель остается в очереди пока не закончится его обслуживание продавцом. В поликлинике пациент находится в кабинете врача, а не в очереди, которая вне кабинета.

Во многих ситуациях, когда выбор заявки на обслуживание в прибор производится из нескольких очередей, очередям могут присваиваться определенные приоритеты. Заявки в приоритетные очереди направляются либо в соответствии с их типами и параметрами, либо потому, что предварительно они прошли различные этапы обслуживания. Если заявка поступает в пустую приоритетную очередь, а в это время прибор обслуживает заявку из очереди с низшим приоритетом, то возможны два варианта обслуживания вновь поступившей заявки. В одном варианте поступившая заявка прерывает обслуживание заявки из очереди с низшим приоритетом, в другом –

заявка ждет окончания этого обслуживания и только после этого поступает на обслуживание. В первом варианте приоритет поступившей заявки называется абсолютным, а во втором – относительным. По окончании обслуживания заявки в любом случае прибор берет на обслуживание следующую заявку из очереди с наивысшим приоритетом.

В СМО возможны самые разнообразные комбинации количеств и типов приоритетов, типов и длин очередей, дисциплин выбора заявок из очередей на обслуживание. В дисциплине обслуживания также можно отразить особенности индивидуального поведения заявок: уклонение от обслуживания, переход из одной очереди в другую, обманные действия и т. п.

Самое главное при описании дисциплины обслуживания это то, что дисциплина должна полностью, и однозначно определять последовательность происходящих в системе действий и событий. Именно дисциплина обслуживания определяет, какие события могут происходить в СМО, как эти события влияют на прохождение заявок через СМО и работу всех устройств.

Описывать дисциплину можно различным способом. Например, можно описать все возможные варианты прохождение заявки через СМО. Другой подход к описанию дисциплины состоит в том, чтобы описать в деталях взаимодействие каждого элемента системы с другими элементами во всех возможных ситуациях, в которых он может оказаться. Если предполагается далее исследовать СМО методами имитационного моделирования, то дисциплину обслуживания целесообразно описывать с точки зрения происходящих в ней событий. При этом последовательно перечисляются все типы событий, которые могут происходить в СМО, и описываются все изменения в системе, которые только могут наступать при наступлении события каждого типа в любой возможной ситуации, при любом возможном состоянии системы.

Создание СМО

Обычно процесс моделирования реальной системы выполняется последовательно, от простого к сложному. Сначала создается самая простая модель, в которой учитываются только самые существенные составляющие процесса функционирования моделируемой системы, самые главные процессы взаимодействия дискретных объектов с элементами системы. Если результаты исследования модели не

приводят к достижению поставленной цели, то модель уточняется. Создается более сложная СМО с большим количеством обслуживающих приборов, с учетом большего количества составляющих процесса функционирования, с более сложной дисциплиной обслуживания, с более детальным описанием типов и параметров заявок.

При создании каждой СМО необходимо решить, какие дискретные объекты будем рассматривать при моделировании, какие их особенности необходимо учитывать для достижения цели моделирования. В соответствии с этим принимается решение, заявки каких типов и с какими параметрам будут обслуживаться в СМО, по каким каналам они будут поступать.

Решается также вопрос о том, какие взаимодействия предполагается учитывать при моделировании. Каждому взаимодействию должен соответствовать какой-либо прибор обслуживания. Каждый прибор может моделировать несколько взаимодействий, но в каждый момент времени только одно из них. Другими словами в каждом приборе должна в каждый момент времени обслуживаться только одна заявка, возможно в несколько этапов. Как правило, неважно, каким образом происходит взаимодействие дискретных объектов с элементами системы, как происходит обслуживание. Учитывается только, когда начинается каждое взаимодействие, как долго оно продолжается и когда заканчивается. Неважно, что именно происходит и как, а важно, когда и как долго это происходит. Часто важен также результат взаимодействия дискретных объектов с элементами системы, т. е. какие изменения произошли в заявках и приборах, как изменились их параметры, как это повлияло на характер работы всей системы.

И, наконец, проанализировав процесс функционирования моделируемой системы надо описать дисциплину обслуживания. При этом необходимо определить и обозначить все параметры, как дисциплины обслуживания, так и самой СМО, а также указать возможные значения этих параметров.

Если исследование СМО предполагается методами имитационного моделирования, то выбор законов, определяющих входные потоки заявок и длительности обслуживаний в приборах можно не уточнять. Достаточно только определиться, какие законы необходимы, и возможно выбрать их обозначения и параметры. При написании программы имитационного моделирования для каждого закона должна быть создана отдельная процедура, реализующая алгоритм вычисления длительности интервала времени, определяемого этим законом. При проведении

экспериментов на имитационной модели эти процедуры можно изменять или полностью заменять на другие, не нарушая структуры всей программы моделирования.

Видимо нет и быть не может некоего универсального метода создания СМО, универсального алгоритма описания реальных систем в виде систем массового обслуживания. Поэтому лучше всего созданию и описанию систем массового обслуживания учиться на конкретных примерах.

Пример создания СМО

Рассмотрим для примера создание СМО магазина оптовой торговли. В магазине два продавца и один кассир. Покупатели магазина сначала у любого из продавцов заказывают необходимые им товары, а когда партия товаров подготовлена, оплачивают ее у кассира, возвращаются к продавцу, забирают товары и покидают магазин.

Назовем СМО «Оптовый магазин». Пусть на вход системы поступают однотипные заявки без параметров одним потоком. Каждая заявка будет моделировать одного покупателя или двух и более, если они пришли вместе за одной партией товара. Пусть длительность интервалов времени между поступлениями очередных заявок есть случайная величина, распределенная по закону $F_{вх}$.

Определим три прибора обслуживания. Первые два будут моделировать работу продавцов, третий – кассира. Будем считать, что каждый прибор в любой момент времени может обслуживать только одну заявку. Схема СМО изображена на рис. 1.4.

Зададим следующую дисциплину обслуживания. Каждая поступающая заявка принимается в СМО, если общее количество уже находящихся в системе заявок меньше некоторого значения N. В противном случае заявка теряется. Введение параметра N является простейшей моделью естественного ограничения на количество одновременно находящихся в магазине покупателей. Для более точного моделирования поступления покупателей в магазин можно, например, задать дискретное распределение вероятностей p_i, i=1,2,…,N. Каждая поступающая заявка, застающая в системе i заявок, с вероятностью p_i принимается в систему, и с вероятностью $1–p_i$ не принимается.

Принятая в систему заявка направляется к первому или второму прибору и становится в очередь на обслуживание. Эти очереди назовем первичными. Если прибор, в очередь к которому поступила заявка, свободен, то он мгновенно начинает эту заявку обслуживать. Пусть каждая принятая в СМО заявка направляется в более короткую

первичную очередь. Если эти очереди одинаковые или их нет вообще, то пусть для определенности заявка поступает в очередь к первому прибору.

Рис. 1.4. «Оптовый магазин»

Длительности обслуживаний заявок из первичных очередей в приборах пусть определяются законами F_{11} и F_{12} соответственно. После обслуживания в одном из первых двух приборах заявка из первичной очереди поступает в третий прибор. Если он свободен, заявка мгновенно начинает обслуживаться. Если прибор занят, заявка становится в очередь вслед за другими заявками, которые ранее прошли обслуживание в приборах 1 и 2. Третий прибор после окончания обслуживания заявки по закону F_3 мгновенно берет на обслуживание первую заявку из очереди, либо переходит в состояние ожидания, если очередь пуста.

После обслуживания в третьем приборе заявка мгновенно возвращается в тот прибор, в котором она прошла первое обслуживание и становится в отдельную вторичную очередь на обслуживание. Если прибор свободен, то заявка мгновенно начинает обслуживаться, а после обслуживания покидает систему. Длительности обслуживания заявок из вторичных очередей определяются законами F_{21} и F_{22}.

Будем считать, что вторичные очереди в первых двух приборах имеют высший относительный приоритет перед первичными очередями. Закончив обслуживание заявки, прибор мгновенно берет на обслуживание очередную заявку из вторичной очереди. Только в том случае, когда вторичной очереди нет, прибор берет заявку из первичной очереди. Если обе очереди отсутствуют, то прибор переходит в состояние ожидания, становится свободным.

Сделаем еще предположение. Будем считать, что обслуживаемая прибором заявка продолжает стоять первой в своей очереди и уходит из очереди только тогда, когда ее обслуживание закончится. Иногда удобнее считать, что обслуживаемая заявка находится в приборе, т. е. покидает очередь в момент начала обслуживания (например, при моделировании очереди в кабинет к врачу). В большинстве случаев это не существенно, где находится обслуживаемая заявка. Как удобнее при имитационном моделировании СМО и обработке результатов моделирования, так обычно и считают.

Другие примеры СМО реальных систем приведены в главе 4.

Классификация СМО

Существует большое количество различных моделей систем массового обслуживания и методов их классификации. Рассмотрим некоторые классы СМО.

Прежде всего, СМО разделяются на *марковские* и *немарковские,* что связано с классификацией процессов, которые описывают функционирование системы. Эти два класса систем массового обслуживания аналогичны, соответственно, линейным и нелинейным системам, динамика которых описывается линейными и нелинейными дифференциальными уравнениями. Естественно, не существует общих аналитических методов исследования нелинейных СМО. Поддаются аналитическому исследованию только частные случаи нелинейных систем массового обслуживания, которые выделяются в отдельные классы полумарковских, линейчатых и других СМО.

По своей структуре СМО делят на *одноканальные* и *многоканальные, однофазные* и *многофазные*. В многоканальных системах обслуживание заявок происходит параллельно в нескольких приборах, а многофазные системы предполагают последовательное обслуживание каждой заявки несколькими приборами. Естественно, существуют и смешанные системы, в которых сочетаются различные варианты и параллельного, и последовательного обслуживания.

В зависимости от того, все ли поступающие в систему заявки принимаются на обслуживание, СМО подразделяют на *системы без потерь,* в которых обслуживаются все поступающие в систему заявки, и *системы с потерями*, в которых при определенных условиях заявки не принимаются на обслуживание, т. е. «теряются».

СМО называются *замкнутыми*, если обслуженные в системе заявки снова и снова возвращаются на обслуживание в этой системе. В *разомкнутых* системах заявки после обслуживания покидают систему.

Системы массового обслуживания, состоящие из большого количества взаимосвязанных обслуживающих приборов, часто называют *сетями СМО*, в отличие от одиночных СМО с простой структурой.

Задачи теории массового обслуживания

Теория массового обслуживания (ТМО), это область прикладной математики, использующая методы теории случайных процессов для исследования процессов функционирования систем на моделях типа СМО. Первоначально специфические вопросы, приведшие к

возникновению теории, появились в связи с эксплуатацией телефонных сетей. Затем оказалось, что подобные задачи возникают при организации торговли (расчет числа магазинов, продавцов в магазинах, запасов товаров и т. д.), целесообразной эксплуатации оборудования, расчете пропускной способности дорог, мостов, аэропортов, компьютерных сетей и т. д.

Классификация целевых назначений теории массового обслуживания основана на выделении в ее структуре различных классов задач и соответственно областей применения получаемых результатов. Можно выделить следующие классы задач:

• задачи анализа поведения систем;

• статистические задачи;

• операционные задачи.

Эти задачи возникают практически в каждой области приложений теории массового обслуживания.

Задачи анализа поведения систем

Цель рассмотрения задач такого рода заключается в том, чтобы с помощью математических моделей, более или менее детально отражающих свойства реальных систем обслуживания, выявить операционные характеристики, определяющие поведение этих систем в процессе их функционирования. К числу основных операционных характеристик любой системы массового обслуживания относятся, в частности, следующие:

• *длина очереди в момент времени t,* т. е. число требований, находящихся в очереди или в узле обслуживания в момент времени *t* (некоторые авторы определяют длину очереди как число требований, ожидающих обслуживания, т. е. не включают в неё требования, обслуживание которых уже начато);

• *продолжительность ожидания начала обслуживания заявкой, поступившей в момент времени t*;

• *продолжительность периода занятости СМО, начавшегося в тот момент, когда в системе было i заявок*;

• *продолжительность n-го интервала простоя системы,* т. е. длина интервала времени, в течение которого система в *n*-й раз оказывается незанятой (ясно, что в системах обслуживания периоды незанятости сменяются периодами занятости).

Наряду с указанными операционными характеристиками, в исследованиях по теории массового обслуживания используются и различные модификации этих и других основных характеристик.

Поскольку входные потоки и потоки обслуживаний в СМО характеризуются потоками случайных событий, то большинство характе- ристик СМО представляют собой случайные процессы и случайные величины. Определение свойств распределений и моментов этих случайных величин и процессов и есть та основная цель, которую преследуют при анализе поведения систем массового обслуживания.

Многие системы массового обслуживания обладают тем свойством, что по истечении определенного времени их поведение в некотором смысле стабилизируется, т. е. перестает меняться с течением времени. В этом случае говорят, что процесс ее функционирования является стационарным, или равновесным. Условия существования стационарного режима, условия, при которых системы переходят в стационарное состояние, представляют значительный интерес.

На практике удобнее оперировать простыми и легко измеримыми показателями, нежели сложными формулами для распределений вероятностей и моментов. Выбор показателей такого рода, разумеется, зависит от характера исследуемой системы и целей анализа. Например, можно исследовать средние длины очередей, интенсивности работы приборов, среднее время пребывания заявки в системе, вероятность занятости прибора в стационарном режиме работы системы и т. д. В этой связи можно подчеркнуть, что наиболее часто используемым показателем является степень загруженности СМО, определяемая коэффициентом нагрузки системы. Этот коэффициент вычисляется как отношение приведенной интенсивности поступления заявок к приведенной интенсивности обслуживания.

Статистические задачи

Статистическое исследование является неотъемлемой частью разработки математической модели реальной системы. В общем виде модель может существовать сама по себе, но приведение ее в количественное соответствие с конкретной системой обслуживания достигается путем статистического анализа эмпирических данных, оценивания фигурирующих в модели параметров и проверок исходных гипотез.

Параметрами системы, по существу, являются параметры, ассоциированные с процессом поступления заявок и механизмом обслуживания (либо некоторые функции этих параметров). Это утверждение нетрудно конкретизировать. Пусть заявки поступают соответственно в моменты времени $t_1, t_2, \ldots$ Величина $\Delta t_n = t_n - t_{n-1}$ есть длина временного интервала между моментами поступления $(n - 1)$-й и

n-й заявками. Обозначим через $\Delta\tau_n$ продолжительность обслуживания n-й заявки C_n. (В случае, когда заявки поступают группами, эти определения необходимо надлежащим образом модифицировать; кроме того, следует ввести понятие распределения применительно к размеру группы.) Пусть $A(x) = P\{\Delta t < x\}$ $(x > 0)$ функция распределения длительности интервалов Δt_n и $B(x) = P\{\Delta\tau < x\}$ $(x > 0)$ функция распределения $\Delta\tau_n$. (Для упрощения обозначений индекс n, которым следовало бы снабдить величины $A(x)$ и $B(x)$, опущен.) Таким образом, параметрами системы являются параметры функций $A(x)$ и $B(x)$, распределения вероятностей некоторых других величин (например, в случае групповых поступлений – размера групп), а также физические характеристики системы такие, как количество обслуживающих приборов, число очередей, вместимость пространства, отведенного для заявок, ожидающих обслуживания и т. д. Таким образом, статистические задачи, возникающие при исследовании систем обслуживания, связаны с оценкой параметров основных процессов, протекающих в системе.

Операционные задачи

Это задачи поиска наилучшего в некотором смысле варианта организации СМО среди множества других возможных вариантов. Существование в теории массового обслуживания таких задач операционной направленности позволяет считать эту теорию одним из разделов исследования операций. Операционные задачи возникают как при проектировании систем, так и при организации работы реально существующих систему и заключаются, например, в выборе наилучшей структуры системы и в таком подборе параметров системы, при котором её функционирование в определенном смысле является наилучшим.

Операционные задачи наиболее сложные в теории массового обслуживания, поскольку обычно включают в себя как задачи анализа поведения, так и статистические задачи. Достаточно сложны аналитические методы решения таких задач, а методы имитационного моделирования предполагают проведения большого числа экспериментов на модели.

Методы решения задач ТМО

В теории массового обслуживания, как и во всякой теории, используются аналитические и численные методы, а также имитационное моделирование. Аналитические и численные методы применяются в основном для решения марковских СМО достаточно

простой структуры и отдельных задач теоретического исследования немарковстих СМО. При этом используется аппарат теории вероятностей, теории случайных процессов, методы решения дифференциальных и разностных уравнений. Ввиду сложности применяемых аналитических методов и ограниченности их использования далее не будем их рассматривать. Основное внимание уделим имитационному моделированию, как универсальному методу исследования систем массового обслуживания.

Имитационное моделирование – это «грубый силовой прием» решения задач анализа и операционных задач теории массового обслуживания, позволяющий исследовать системы любой сложности. Возможности имитационного моделирования ограничены лишь мощностями вычислительной техники, на которой производится моделирование, и квалификацией, опытом и талантом специалистов, создающих имитационные модели. Далее рассматривается одна из возможных технологий создания имитационных моделей СМО любой сложности, основанная на применении универсальных языков программирования (Паскаль, Си и т. д.).

Глава 2

Имитационное моделирование

Основные понятия и определения

«Имитационное моделирование есть процесс конструирования модели системы и постановки экспериментов на этой модели с целью либо понять поведение системы, либо построить теории и гипотезы, которые бы объясняли поведение системы как в настоящем времени, так и в будущем». Это определение дал Шеннон в своей книге «Имитационное моделирование систем – искусство и наука»[1].

Что следует из этого определения? Во-первых, при имитационном моделировании исследуются процессы функционирования систем как реальных, так и проектируемых. Во-вторых, имитационные модели предполагают проведение на них экспериментов, что вполне естественно при исследовании поведения.

Далее под имитационным моделированием будем понимать метод исследования процессов функционирования систем на имитационных моделях, выполненных в виде компьютерных программ. При этом постановка различных экспериментов на модели будет означать не что иное, как выполнение программы имитационного моделирования при различных исходных данных. Результатами каждого эксперимента будут численные данные, выдаваемые программой в той или иной форме. Мы не будем касаться общих проблем имитационного моделирования произвольных систем, а рассмотрим только вопросы имитации функционирования систем массового обслуживания как математических моделей реальных или проектируемых систем.

Процесс функционирования всякой системы есть изменение ее состояния с течением времени. Если состояние системы определяется значениями некоторых переменных s_1, s_2, ..., s_n, которые составляют вектор состояния системы $S = [s_1, s_2, ..., s_n]$, то процесс ее работы можно полностью описать изменениями значений этих переменных во времени, т. е. функциями времени $s_1(t)$, $s_2(t)$, ..., $s_n(t)$. Один из методов исследования реальных, действующих систем, состоит в том, что на некотором интервале времени длины T наблюдается работа системы. В определенные последовательные моменты времени t_0, t_1, ..., t_k на этом интервале наблюдения измеряются переменные ее состояния, т. е. в эти моменты определяются значения S_0, S_1, ..., S_k вектора состояния системы. Затем полученные результаты t_0, t_1, ..., t_k и S_0, S_1, ..., S_k

обрабатываются и по ним вычисляются численные характеристики работы системы.

При имитационном моделировании по предлагаемой в данной работе технологии происходит почти то же самое, только наблюдается не сам процесс функционирования реальной системы. Сначала создается математическая модель реальной системы в виде системы массового обслуживания, а затем имитируется на компьютере процесс функционирования этой СМО. Программа имитационного моделирования последовательно по определенному алгоритму вычисляет значения моментов времени t_0, t_1, ..., t_k и значения вектора состояния системы в эти моменты времени S_0, S_1, ..., S_k. Процесс имитации представляет собой циклический процесс, на каждом этапе которого вычисляется очередной момент t_i и состояние системы S_i в этот момент, а также выполняется необходимая предварительная обработка и накопление этих данных. По завершении процесса имитации в программе выполняется окончательная обработка полученных данных, т. е. вычисляются численные оценки характеристик работы системы, и результаты выдаются в том или ином виде. Точность полученных при моделировании оценок зависит, естественно, от алгоритмов вычисления моментов времени t_0, t_1, ..., t_k и состояний системы в эти моменты, а алгоритмы, в свою очередь, от типов и особенностей самих процессов функционирования систем.

Основная проблема имитационного моделирования состоит именно в разработке алгоритмов вычисления t_0, t_1, ..., t_k и S_0, S_1, ..., S_k, а также в реализации этих алгоритмов, поскольку методы обработки результатов моделирования и получения оценок процессов функционирования систем достаточно хорошо известны в математической статистике и теории эксперимента. Существует много различных технологий, приемов и методов создания программ имитационного моделирования систем массового обслуживания как на универсальных, так и на специально созданных для этого языках программирования. Мы рассмотрим только одну из технологий создания программ имитационного моделирования на любом из универсальных языков программирования.

Одна из особенностей структуры программы моделирования, созданной по рассматриваемой технологии, состоит в разделении программы на две части: имитационную модель и программу эксперимента. Для относительно простых моделей каждая часть может представлять собой комплекс процедур и функций, написанных на

выбранном для моделирования языке программирования и выполняющих определенные действия.

Имитационная модель – это основная часть программы моделирования конкретной СМО. Эту часть составляют процедуры вычисления моментов времени t_0, t_1, ..., t_k и формирования состояний системы, т. е. вычисления переменных состояния СМО в эти моменты времени. Процедуры имитационной модели не изменяются при выполнении любых экспериментов.

В программу эксперимента входят процедуры, которые задают исходные данные для каждого эксперимента, управляют экспериментом, выполняют обработку результатов моделирования, вычисляют оценки характеристик работы системы и обеспечивают интерфейс с пользователем. Эта часть программы не постоянна. Отдельные ее компоненты создаются с учетом целей и задач конкретного эксперимента.

Этапы моделирования СМО

Под имитационным моделированием любой системы массового обслуживания обычно понимают весь процесс до получения численных характеристик работы системы. В рамках рассматриваемой технологии в этом процессе можно выделить следующие основные этапы.

Прежде всего необходимо решить, на каком языке программирования будет создаваться программа имитационного моделирования. Это самый простой этап создания модели. Никаких особых требований к языку программирования технология не предусматривает. Подойдет любой из универсальных языков, например, «Си», который далее и будем использовать во всех приведенных ниже примерах.

Затем надо подобрать идентификаторы переменных и констант имитационной модели, которые далее будут определены в этой главе при описании структуры программы моделирования. Также надо выбрать переменные, массивы, структуры, которые будут использоваться в программе для описания параметров СМО, определить их типы и возможные значения. Необходимо выбрать идентификаторы, типы и аргументы всех функций, которые в программе будут моделировать длительности отдельных процессов. Это процессы обслуживания заявок в приборах, процессы ожидания заявок и внешних воздействий, т. е. те процессы, длительности которых определены соответствующими законами или алгоритмами, заданными или только

обозначенными при создании СМО. В программе моделирования каждому закону должна соответствовать некоторая функция, при обращении к которой она должна возвращать длительность процесса. Далее при создании программы моделирования эти функции должны быть полностью определены, а также должны быть определены все параметры соответствующих законов и алгоритмов.

На следующем этапе проводится анализ СМО. При этом решаются, по крайней мере, две главные задачи. Первая состоит в описании состояния СМО, т. е. в выборе переменных вектора состояния системы s_1, s_2, ..., s_n, значения которых в любой момент времени работы программы моделирования полностью и однозначно определяли бы состояние моделируемой системы. Целесообразно сразу структурировать эти переменные, описать на языке программирования идентификаторы и типы скалярных переменных, массивов, структур, определить возможные значения или диапазоны изменения всех переменных состояния.

Вторая задача заключается в соответствующем описании процесса функционирования системы, т. е. в таком описании процесса изменения ее состояния во времени, которое удобно для организации моделирования моментов времени t_0, t_1, ..., t_k наступления событий и вычисления значений переменных состояния s_1, s_2, ..., s_n.

Далее необходимо написать и отладить имитационную модель как основную часть программы моделирования. Эта часть программы создается по определенной схеме, единой для имитационных моделей любых СМО. Имитационная модель состоит из главной функции программы, функций, в которых моделируются моменты t_0, t_1, ..., t_k и формируются состояния системы в эти моменты времени, а также из ряда других вспомогательных процедур и функций.

Последний этап имитационного моделирования состоит в планировании конкретных экспериментов на имитационной модели, написании соответствующей программы эксперимента или отдельных ее блоков и проведения необходимых экспериментов с целью получения оценок характеристик моделируемой СМО. Поскольку моделирование системы носит вероятностный характер, то, очевидно, для получения достоверных оценок характеристик работы системы необходимо проведение серий экспериментов при неизменных параметрах моделируемой системы.

Рассмотрим более подробно основные этапа моделирования и начнем с описания состояния системы.

Вектор состояния СМО

Состояние любой произвольной СМО можно определить, как уже отмечалось, значениями некоторых переменных s_1, s_2, ... ,s_n, которые будем называть переменными состояния, а их совокупность $S = [s_1, s_2, ..., s_n]$ – вектором состояния СМО.

Задача описания состояния системы состоит в том, какие величины выбрать в качестве переменных состояния, какие значения эти переменные могут принимать и какое смысловое содержание этих значений. Эта задача не имеет однозначного решения. Для каждой системы массового обслуживания можно выбрать разные величины в качестве переменных состояния. При этом необходимо учитывать следующее.

Во-первых, значения этих переменных должны полностью определять любое возможное состояние системы в любой момент времени ее функционирования. При этом состояние должно определяться действительно полностью, не зависимо от того, какие эксперименты предполагается провести и какие характеристики системы оценить по результатам этих экспериментов.

Во-вторых, какое бы событие ни произошло в системе, значений переменных состояния должно быть достаточно для однозначного определения нового значения вектора состояния. Надо, чтобы по значениям переменных состояния до момента наступления любого события всегда можно было вычислить значения этих переменных после этого события.

В-третьих, количество компонент вектора состояния особого значения не имеет. Не обязательно, чтобы переменных состояния было как можно меньше. Переменные надо выбирать так, как удобно для написания имитационной модели и программы эксперимента. Если по значениям одних переменных состояния можно вычислить значения других, то от этих других не следует обязательно отказываться. Их можно и нужно оставить в векторе состояния, если они упрощают программу моделирования или удобны для вычисления характеристик СМО.

При выборе переменных состояния системы необходимо следить за тем, чтобы состояние каждого элемента системы было учтено в векторе состояния. Состояния очередей обязательно определяются количествами заявок в них. Поэтому в векторе состояния должны быть соответствующие переменные целого типа со значениями количеств заявок в очередях или, иначе говоря, со значениями длин очередей.

Иногда этого недостаточно, и необходимо учитывать, какие именно заявки стоят в очередях. В этом случае можно ввести ограничение на длину очереди, т. е. предположить, что длина очереди не может быть больше некоторого значения N. Тогда для описания состояния очереди можно использовать, например, вектор $[r_0, r_1, \ldots, r_N]$, в котором каждая компонента определяет состояние соответствующего места в очереди. Место может быть свободно или занято заявкой определенного типа или с определенными свойствами.

Для описания состояний приборов обычно используются целочисленные переменные, значения которых определяют, занят прибор или свободен, а если занят, то какой вид обслуживания выполняется в данный момент. В простейших случаях, когда прибор обслуживает заявки без параметров и только из одной очереди, то состояние прибора может определяться состоянием этой очереди. Если очередь пуста, то прибор свободен, если в очереди есть заявки, то прибор обслуживает первую из них.

Вектор состояния для СМО «Оптовый магазин»

Система массового обслуживания «Оптовый магазин» достаточно подробно описана в первой главе. Для описания состояния СМО введем следующие переменные.

```
int k1[2];//количество заявок в первичных
очередях
int k2[2];//количество заявок во вторичных
очередях
```

Первому прибору соответствует первый элемент массивов с индексом 0, второму – с индексом 1. Значения этих целочисленных переменных могут изменяться от нуля до максимально возможного количества N заявок.

```
int s[2]; //состояния первого и второго приборов
```

Значения этих переменных будут определять состояния приборов следующим образом: 0 – прибор свободен и находится в состоянии ожидания; 1 – в приборе обслуживается заявка из первичной очереди; 2 – обслуживается заявка из вторичной очереди.

Для описания состояния третьего прибора будем использовать элементы массива

```
int r[N]; //состояние очереди к третьему прибору
```

Каждый элемент *r*[*i*] определяет состояние *i*-го места в очереди. Если *i*-е место в очереди свободно, переменная имеет нулевое значение. Если место занято заявкой, переменная *r*[*i*] имеет значение номера прибора, в котором заявка прошла первичное обслуживание. Состояние третьего прибора при этом полностью определяется значением *r*[0]. Если значение равно нулю, то прибор свободен, т. к. свободно самое первое место в очереди и свободны все остальные места. Если равно 1 или 2, то в приборе обслуживает заявка, поступившая, соответственно, из первого или из второго прибора обслуживания.

На первый взгляд может показаться, что вместо массива `r` для описания состояния очереди достаточно одной переменной:

```
int k3;   //количество заявок в очереди к
третьему прибору.
```

Но эта переменная не определит полностью состояние третьего прибора и очереди к нему, поскольку не позволяет определить, в какое состояние перейдет система после окончания обслуживания заявки в третьем приборе. Для этого надо знать, в каком приборе обслуженная заявка прошла первичное обслуживание, чтобы направить её на вторичное обслуживание в этот же прибор.

Переменную `k3` можно включить в вектор состояния как, впрочем, и переменную

```
int k0;        //общего количества заявок в
системе
```

Эти переменные будут необходимы и полезны как при написании программы моделирования, так и для обработки результатов моделирования.

Выбранные 4 массива и 2 скалярные переменные образуют один из возможных векторов состояний для СМО «Оптовый магазин». Состояние системы можно также описывать, например, некоторой матрицей целочисленных переменных, состоящей из *N* строк. Каждая строка в этой матрице будет соответствовать одной находящейся в системе заявке, а элементы строки (столбцы матрицы) некоторым параметрам заявки. Например, если в системе находятся *K* заявок, то *K* строк матрицы будут содержать информацию об этих заявках, а переменные остальных строк будут иметь, например, нулевые значения. В столбцах матрицы можно сохранять информацию, например, о порядковых номерах находящихся в системе заявок, о номерах очередей, в которых в данный момент находятся заявки, о местах, занимаемых заявками в очередях и т. д.

Если для описания переменных состояния системы использовать не матрицу с целочисленными элементами, а массив структур из N компонент, то в каждой структуре, соответствующей одной находящейся в системе заявки, можно сохранить больше информации и не только целочисленной. Можно, например, хранить информацию о времени поступления заявки систем, о временах пребывания заявки в очередях и приборах и т. д. Такое описание вектора состояния системы не всегда удобно при написании программы моделирования, но может быть весьма полезен для обработки результатов экспериментов.

Другие примеры описания состояний систем массового обслуживания можно посмотреть в главе 4

Процесс функционирования СМО

Для того чтобы описать технологию моделирования моментов перехода системы из состояния в состояние, проанализируем процесс функционирования моделируемой системы и представим его соответствующим образом.

Работа СМО – дискретно-непрерывный процесс

Работу всякой СМО можно представить как процесс ее перехода из одного состояния в другое, в общем случае в случайные моменты времени t_0, t_1, ..., t_k, когда в СМО происходят события, предусмотренные дисциплиной обслуживания. Каждое событие приводит к изменениям каких-либо переменных состояния системы. К таким событиям относятся поступления заявок, различные внешние воздействия на систему, факты постановок заявок в очереди, взятие заявок на обслуживания, окончания обслуживаний, переключения режимов работы приборов и т. д. Какие события могут происходить в конкретной СМО определяется, как уже отмечалось, дисциплиной обслуживания.

Допустим, что моделируемая система находится в некотором состоянии S_0. Она продолжает оставаться в этом состоянии, пока не произойдет одно из событий. Считается, что до наступления этого события в некоторый момент t_0 в системе не происходит ничего существенного. В момент времени t_0 система скачком переходит в другое состояние S_1. В какое именно зависит от того, каково состояние S_0 и какое событие произошло в момент t_0. В состоянии S_1 система продолжает оставаться до тех пор, пока не наступит следующее событие в некоторый другой момент времени t_1. Это событие переводит систему в другое состояние S_2, в общем случае отличное от S_1. По-прежнему считается, что на интервале времени от t_0 до t_1 в системе ничего

существенного не произошло. Из S_2 система переходит в состояние S_3 под воздействием события в момент времени t_2 и т. д.

Таким образом, процесс функционирования любой СМО есть дискретно-непрерывный процесс, т. е. процесс с дискретным множеством значений (состояний системы) и непрерывным изменением времени. Состояние случайного процесса скачком изменяется в некоторые моменты времени t_0, t_1, …, t_k наступления событий и остаются неизменными на интервалах времени между этими моментами как это изображено на рис. 2.1.

Рис. 2.1. Процесс функционирования СМО

Длительности интервалов между моментами очередных событий есть в общем случае непрерывные случайные величины с неизвестными законами распределения. Часто именно эти законы распределения являются объектами исследования. Если бы эти законы распределения были известны, то для имитации интервалов между моментами t_0, t_1, …, t_k, а, следовательно, и самих этих моментов, можно было бы воспользоваться известными методами моделирования значений случайных величин с заданными законами распределения. Но законы распределения длительностей интервалов между моментами t_0, t_1, …, t_k, неизвестны. Следовательно, необходим более детальный анализ процесса функционирования СМО для его успешного моделирования.

Основные и вспомогательные события

Прежде всего, проанализируем происходящие в СМО события. Все события, возникающие в любой системе массового обслуживания, можно поделить на основные и сопутствующие, или вспомогательные. Основные события – это такие события, которые являются первопричиной изменения состояний системы. Вместе с каждым основным событием в СМО, как правило, наступает несколько вспомогательных событий, как следствие этого основного события. Из всех событий, наступающих в момент изменения состояния СМО, одно и только одно является основным, а остальные вспомогательными. Каждое сопутствующее событие является следствием основного события и наступает одновременно с ним.

Например, на вход СМО поступает заявка. Это событие не влечет за собой других событий, если заявка не принимается в систему, и

состояние системы при этом не изменяется. Если же заявка принимается, то, как следствие этого, наступает вспомогательное событие – заявка становится в очередь на обслуживание. Если обслуживающий прибор в это время свободен, то заявка из очереди сразу поступает на обслуживание в этот прибор. Взятие заявки на обслуживание – это то же событие, которое наступило как следствие поступления заявки в СМО и одновременно с этим событием.

Другой пример. Прибор закончил обслуживание заявки. Это событие в зависимости от дисциплины обслуживания влечет за собой достаточно много вспомогательных событий. Обслуженная заявка освобождает прибор. Обслуженная заявка становится в очередь к следующему прибору или поступает на обслуживание в этот прибор, если он в данный момент свободен. Обслуженная заявка покидает СМО, если так предусмотрено дисциплиной обслуживания. Прибор, обслуживший заявку, переходит в состояние ожидания или берет на обслуживание следующую заявку. Эта следующая заявка могла ожидать обслуживания не в очереди, а в предыдущем приборе, не давая ему возможность обслуживать свои заявки. В этом случае заявка освобождает этот прибор, и он меняет свое состояние: остается свободным или берет на обслуживание заявку. И так далее.

Для создания программ имитационного моделирования любой СМО по рассматриваемой технологии необходимо из всего множества событий выделить основные события. Если рассматривать работу СМО на большем интервале времени, то таких событий может быть очень много. Поэтому важно не только выявить основные события, но и определить типы, а главное, последовательности их наступления, т. е. определить потоки этих основных событий.

Локальные процессы и их последовательности

Для того чтобы выявить основные события, их типы и потоки, представим работу любой СМО как совокупность некоторого множества отдельных, элементарных, «коротких» процессов, длительность каждого из которых определена некоторым законом или алгоритмом, заданным при описании СМО. Именно такие элементарные процессы, длительности которых определяются заданными законами, далее будем называть *локальными процессами*.

Одновременно в системе могу происходить несколько различных локальных процессов. Они начинаются в некоторые моменты работы системы, какое-то время продолжаются согласно законам, которые их определяют, заканчиваются, затем начинаются другие и т. д. При этом

почти не имеет значения, что представляет собой каждый локальный процесс, что конкретно при этом происходит. Важно, какое событие в системе является началом локального процесса, как долго процесс продолжается, что происходит при его окончании.

Выявить локальные процессы, происходящие в конкретной СМО, не представляет труда, поскольку при описании СМО задаются законы, определяющие их длительность. Эти законы и дисциплина обслуживания определяют типы локальных процессов. Длительности локальных процессов нередко задаются постоянными или зависящими от текущего состояния системы, но, как правило, они предполагаются случайными. В общем случае, законы, определяющие длительности локальных процессов, можно считать законами распределения случайных величин, а соответствующие им алгоритмы, методами моделирования случайных величин.

Для целей имитационного моделирования важно не только выявить локальные процессы. Необходимо выявить такие последовательности локальных процессов, в которых каждый следующий процесс начинается не раньше, чем закончится предыдущий. Обычно такие последовательности образуют процессы, которые связаны с определенным элементом системы.

К локальным процессам в любой СМО относятся, прежде всего, процессы, связанные с потоками заявок в систему. Действительно, каждый входной поток заявок в первую очередь задается законами (или законом), определяющими длительности интервалов времени между поступлениями очередных заявок. Следовательно, процесс поступления любой заявки, или процесс ожидания поступления очередной заявки, является локальным процессом. Если в системе заданы потоки внешних воздействий, то процессы ожидания этих внешних воздействий, естественно, тоже являются локальными процессами.

Каждый входной поток заявок или внешних воздействий, очевидно, образует непрерывную последовательность локальных процессов, каждый из которых есть процесс ожидания очередного поступления заявки или воздействия. Непрерывность здесь означает то, что завершение одного локального процесса является началом следующего, т. е. поступление заявки завершает локальный процесс её ожидания и начинает процесс ожидания следующей.

Каждое обслуживание заявки в каждом приборе является локальным процессом, поскольку длительность такого обслуживания определяется соответствующим законом, заданным при описании СМО. Локальные процессы обслуживания заявок в конкретном приборе также

образуют последовательность, поскольку каждый следующий процесс начинается не раньше, чем закончится предыдущий, если только прибор одновременно обслуживает не более одной заявки. Если в некоторой реальной системе, для которой создается СМО, некоторой элемент одновременно взаимодействует с несколькими дискретными объектами, то целесообразно представить его в виде узла обслуживания с несколькими параллельно работающими приборами, каждый из которых может обслуживать одновременно не более одной заявки. В этом случае локальные процессы обслуживания в каждом из таких приборов образуют свою отдельную последовательность.

Последовательности процессов обслуживания, как правило, не являются непрерывными. После окончания очередного обслуживания прибор может перейти в состояние ожидания, и следующий процесс обслуживания начнется только после поступления заявки. Поэтому, как правило, в последовательностях процессов обслуживания участки непрерывного следования локальных процессов чередуются с периодами отсутствия этих процессов.

При задании СМО часто учитывается ненадежность обслуживающих приборов. Предполагается, что приборы могут выходить из строя, а затем восстанавливаться. При этом задаются законы, определяющие длительности пребывания приборов в рабочем состоянии и в состоянии восстановления. Если эти законы не предполагают однозначной зависимости длительностей периодов работы и ремонта приборов от законов, определяющих длительности каких-либо других процессов, то периоды работы и восстановления следует считать локальными процессами. Можно считать, что локальные процессы работы конкретного прибора образуют свою прерывистую последовательность однотипных процессов, а процессы восстановления – свою. Можно считать иначе. Эти процессы в конкретном приборе образуют единую непрерывную, чередующуюся последовательность разнотипных, локальных процессов, в которой за процессом работы сразу следует процесс восстановления и наоборот.

Как правило, такие процессы, как ожидания прибором поступления заявки, пребывание заявки в очереди или, иными словами, ожидания заявкой начала обслуживания, нельзя назвать локальными процессами, поскольку длительности таких процессов не определяются какими-либо конкретными законами, а зависят от законов поступления и обслуживания заявок.

Поскольку локальные процессы заканчиваются в общем случае через время, определяемое некоторым заданным законом, то каждый

факт окончания локального процесса является основным событием. Действительно, любой локальный процесс заканчивается независимо от того, какие события наступают одновременно с его окончанием, т. е. факт окончания локального процесса не является следствием какого-либо одномоментного с ними события.

Далее основными событиями в любой СМО будем называть окончания локальных процессов, а потоками основных событий – окончания локальных процессов в последовательностях. Таким образом, потоков основных событий в любой СМО столько, сколько есть в ней последовательностей локальных процессов. Каждая последовательность локальных процессов порождает один поток основных событий.

Процесс моделирования

Рассмотрим теперь вопросы следующего этапа моделирования. Этот этап, как уже отмечалось, состоит в написании функций или процедур имитационной модели, которые являются основой программы имитационного моделирования.

Но прежде чем рассмотрим структуру программы моделирования, логику и принципы написания процедур имитационной модели, выполним моделирование конкретной системы массового обслуживание без компьютера и программирования. Возьмем для примера снова систему массового обслуживания «Оптовый магазин», которая описана в первой главе.

Моделирование без программирования

При создании СМО «Оптовый магазин» было решено задавать один закон $F_{вх}$ для моделирования длительностей интервалов времени между моментами поступления очередных заявок (закон поступления заявок), четыре закона обслуживания F_{11}, F_{12}, F_{21} и F_{22} в первом и втором приборах (первый индекс – номер очереди, из которой обслуживается заявка, второй – номер прибора) и один закон F_3 обслуживания всех заявок в третьем приборе. Следовательно, процесс функционирования СМО формируется шестью типами локальных процессов, и процессы этих типов образуют четыре последовательности. Одну последовательность образуют процессы ожидания поступления заявок в систему. Каждый процесс этого типа начинается сразу после окончания предыдущего. Поскольку во всех приборах в каждый момент времени обслуживается только одна заявка, процессы обслуживания в приборах образуют еще три

последовательности, причем в первых двух приборах эти последовательности образуют процессы двух типов.

Результаты моделирования будем отображать на своеобразной диаграмме, которая изображена на рис. 2.2. В нижней части диаграммы горизонтальная прямая изображает ось времени моделирования t. На этой оси будем отмечать моменты t_0, t_1, …, t_k, наступления событий, изменяющих состояние системы. Вертикальная прямая слева пусть обозначает момент начала процесса моделирования t_0. Над осью времени параллельно ей 4 прямые изображают оси времени τ_i, соответствующие четырем последовательностям локальных процессов, определенных для данной СМО. Ось τ_0 соответствует локальным процессам ожидания поступления заявок. Следующие оси τ_1, τ_2 и τ_3 – последовательностям локальных процессов обслуживания, соответственно, в первом, во втором и в третьем приборах. На этих осях дугами будем изображать длительности локальные процессов. Законы, определяющие длительности, конкретизировать не будем, и поэтому все дуги будем изображать произвольной длины. Левые концы дуг будут соответствовать моментам, когда начинаются локальные процессы, а правые концы – моментам окончания локальных процессов, т. е. моментам наступления основных событий.

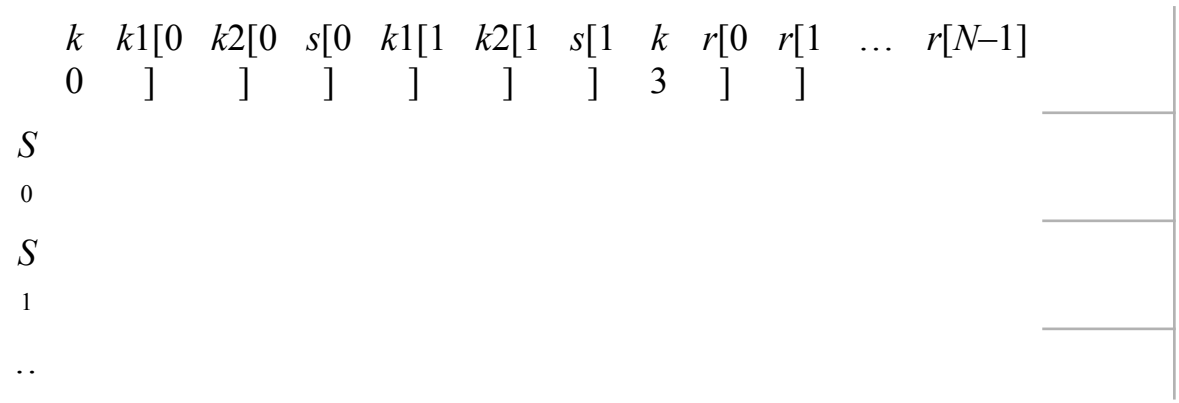

Рис. 2.2. Диаграмма процесса моделирования

Состояния, в которых система побывает при моделировании, будем записывать в таблицу, которая также представлена на рис. 2.2. Строки таблицы соответствуют состояниям, а столбцы – компонентам вектора состояния, которые ранее уже были выбраны и описаны на зыке «С». Напомним: $k0$ – общее количество заявок в системе; $k1[i]$, $k2[i]$– количество заявок в первичных и вторичных очередях первого ($i = 0$) и второго ($i = 1$) приборов; и $s[i]$ – состояния первого и второго приборов;

$k3$ – количество заявок в очереди к третьему прибору; $r[j]$ – состояния мест для заявок в очереди к третьему прибору.

Для того чтобы начать процесс моделирования, зададим некоторое начальное состояние S_0, в котором система находится в момент начала моделирования. Пусть, например, это состояние, при котором в системе нет ни одной заявки.

При состоянии S_0 в системе может происходить только один локальный процесс – ожидание поступления первой заявки. Пусть для определенности он заканчивается в момент начала моделирования t_0, т. е. в момент начала моделирования в систему поступает первая заявка. По дисциплине обслуживания эта заявка сразу поступает в первичную очередь к первому прибору и начинается процесс ее обслуживания. Длительность этого локального процесса определяется законом F_{11}. На диаграмме на рис. 2.3 длительность отмечена дугой на оси τ_1, левый конец которой совпадает с t_0. Длину дуги рисуем произвольно, предполагая, что длину определили по закону F_{11}.

В момент t_0 начинается еще один локальный процесс – процесс ожидания поступления следующей заявки. Этот процесс помечен на диаграмме дугой произвольной длины на оси τ_0, В действительности же длина этой дуги должна быть определена по закону $F_{вх.}$

	$k0$	$k1[0]$	$k2[0]$	$s[0]$	$k1[1]$	$k2[1]$	$s[1]$	$k3$	$r[0]$	$r[1]$	…	$r[N–1]$
S_0	0	0	0	0	0	0	0	0	0	0	…	0
S_1	1	1	0	1	0	0	0	0	0	0	…	0

Рис. 2.3

Система при поступлении заявки в момент t_0 переходит в состояние S_1 и продолжает оставаться в этом состоянии до тех пор, пока не закончится один из происходящих в ней локальных процессов. Иначе говоря, пока в системе не произойдет одно из основных событий, предусмотренных дисциплиной обслуживания. На диаграмме изображен вариант, когда локальный процесс ожидания заявки заканчивается в момент t_1 раньше процесса обслуживания заявки. Таким образом, в момент t_1 в системе происходит событие (приход новой заявки), которое изменяет состояние системы S_1.

Поступившая в момент t_1 вторая заявка по дисциплине обслуживания попадает в первичную очередь ко второму прибору и сразу начинается ее обслуживание. На рис. 2.4 обслуживание во втором приборе изображено дугой произвольной длины на оси τ_2. В момент t_1 начинается также локальный процесс ожидания третьей заявки, который на рис. 2.4 изображен дугой на оси τ_0. Система переходит из состояния S_1 в состояние S_2.

При состоянии S_2 в системе происходят три локальных процесса: ожидание прихода следующей заявки, обслуживание в первом приборе, обслуживание во втором приборе. Соответствующие процессам дуги нарисованы так, что раньше других закончится процесс обслуживания в первом приборе. Событие окончания этого процесса в момент времени t_2 будет тем событием, которое изменит состояние системы S_2 на новое состояние S_3.

По дисциплине обслуживания заявка, после обслуживания в первом приборе, поступает в очередь к третьему прибору и начинает в нем обслуживаться, поскольку прибор в этот момент свободен. Освободившийся первый прибор переходит в свободное состояние, так как ни в одной из двух очередей к нему нет заявок. Таким образом, в момент t_2 система переходит в состояние S_3, которое представлено на рис. 2.5.

	$k0$	$k1[0]$	$k2[0]$	$s[0]$	$k1[1]$	$k2[1]$	$s[1]$	$k3$	$r[0]$	$r[1]$	…	$r[N–1]$	
S_0	0	0	0	0	0	0	0	0	0	0	…	0	
S_1	1	1	0	1	0	0	0	0	0	0	…	0	
S_2	2	1	0	1	1	0	1	0	0	0	…	0	

Рис. 2.4

	$k0$	$k1[0]$	$k2[0]$	$s[0]$	$k1[1]$	$k2[1]$	$s[1]$	$k3$	$r[0]$	$r[1]$	…	$r[N–1]$
S_0	0	0	0	0	0	0	0	0	0	0	…	0
S_1	1	1	0	1	0	0	0	0	0	0	…	0
S_2	2	1	0	1	1	0	1	0	0	0	…	0
S_3	2	0	0	0	1	0	1	1	1	0	…	0

Рис. 2.5

Значение $r[0] = 1$ в стоке S_3 означает, на первом месте в очереди к третьему прибору стоит заявка, которая прошла первичное обслуживание в первом приборе. Нулевые значения в последних столбцах означают, что все места кроме первого в этой очереди свободны.

При новом состоянии S_3 в системе продолжаются процессы ожидания заявки и обслуживания во втором приборе, и начинается новый локальный процесс обслуживания заявки в третьем приборе. Дуга произвольной длины, соответствующая новому процессу, изображена на оси τ_3 на рис. 2.5.

Все дуги нарисованы так, что следующим событием, изменяющим состояние системы, будет приход третьей заявки. В результате в момент t_3 система переходит в состояние S_4, которое изображено на рис. 2.6. По дисциплине обслуживания поступившая заявка попадет в первичную очередь к первому прибору. А поскольку первый прибор в этот момент t_3 свободен, то начнется обслуживание в нем поступившей заявки.

	$k0$	$k1[0]$	$k2[0]$	$s[0]$	$k1[1]$	$k2[1]$	$s[1]$	$k3$	$r[0]$	$r[1]$	…	$r[N–1]$
S_0	0	0	0	0	0	0	0	0	0	0	…	0

S_1	1	1	0	1	0	0	0	0	0	0	…	0
S_2	2	1	0	1	1	0	1	0	0	0	…	0
S_3	2	0	0	0	1	0	1	1	1	0	…	0
S_4	3	1	0	1	1	0	1	1	1	0	…	0

Рис. 2.6

В системе начинаются два новых локальных процесса: ожидание прихода четвертой заявки и процесс первичного обслуживания третьей заявки в первом приборе, а также продолжаются процессы обслуживания во втором и третьем приборах.

На диаграмме на рис. 2.6 дуги, соответствующие всем четырем процессам изображены так, что следующим основным событием, в результате наступления которого система перейдет в состояние S_5, будет приход следующей четвертой заявки.

Поступившая заявка согласно дисциплине обслуживания поступит в первичную очередь к первому прибору и будет ожидать обслуживания. В системе начнется новый локальный процесс ожидания пятой заявки, который изображен соответствующей дугой на оси τ_0 на рис. 2.7, а новое состояние S_5 в таблице на этом рисунке.

	$k0$	$k1[0]$	$k2[0]$	$s[0]$	$k1[1]$	$k2[1]$	$s[1]$	$k3$	$r[0]$	$r[1]$	…	$r[N-1]$
S_0	0	0	0	0	0	0	0	0	0	0	…	0
S_1	1	1	0	1	0	0	0	0	0	0	…	0
S_2	2	1	0	1	1	0	1	0	0	0	…	0

S_3	2	0	0	0	1	0	1	1	1	0	…	0
S_4	3	1	0	1	1	0	1	1	1	0	…	0
S_5	4	2	0	1	1	0	1	1	1	0	…	0
S_6	4	2	0	1	0	0	0	2	1	2		0

Рис. 2.7

Как видно на рис. 2.7, следующим событием, которое изменит состояние S_5, будет окончание обслуживания во втором приборе в момент t_5. При этом обслуженная во втором приборе заявка станет на второе место в очередь к третьему прибору, а второй прибор перейдет в состояние ожидания. Система перейдет в состояние S_6, которое также представлено в таблице на рис 2.7.

Моделирование на диаграмме можно продолжать сколько угодно долго. Если бы длительности локальных процессов выбирались не произвольно, а вычислялись в соответствии с выбранными, конкретными законами, и моделирование продолжалось достаточно долго, то данные о состояниях системы и моментах времени изменения этих состояний было бы достаточно для вычисления оценок любых параметров и характеристик процесса функционирования выбранной СМО.

Какие можно сделать выводы из рассмотренного примера моделирования на диаграмме?

Во-первых, прежде чем начать моделирование, необходимо задать начальное состояние S_0, в котором система пребывает до момента времени начала моделирования, и время начала моделирования t. Необходимо смоделировать или задать длительности локальных процессов, которые происходят при начальном состоянии S_0 с учетом того, что в момент начала моделирования закончится один из этих процессов. Таким образом, должны быть определены моменты наступления основных событий, которые могут изменить состояние S_0, и должно быть известно, какое событие наступает в момент начала моделирования.

Во-вторых, моделирование представляет собой циклический процесс. Каждый цикл состоит в формировании состояния системы, в

моделировании длительности начинающихся при этом состоянии локальных процессов и в определении момента наступления нового состояния.

Рассмотрим теперь вопрос, что необходимо сделать для того, чтобы запрограммировать процесс моделирования, какие необходимы переменные, функции, какова должна быть структура программы.

Модельное время

Поскольку процесс функционирования системы моделируется во времени, причем, как уже отмечалось, последовательно вычисляются моменты наступления событий t_0, t_1, ..., то вполне естественно использовать понятие «модельное время» или «таймер модельного времени» в качестве одного из элементов модели. Для этого в программе должна быть введена специальная переменная t. На приведенных выше диаграммах этой переменной соответствует ось t.

Модельное время – это текущее время работы модели, это то значение времени, до которого на данном этапе работы программы уже выполнена имитация процесса функционирования СМО. До момента t уже всё известно об имитируемой реализации процесса функционирования СМО, т. е. известны моменты изменений состояний системы, которые меньше t, и известны состояния, в которых побывала система до этого момента времени. После момента t реализация процесса функционирования СМО неизвестна и ее еще предстоит имитировать на последующих этапах работы программы моделирования.

В начале моделирования этой переменной присваивают какое-то определенное начальное значение, чаще всего нулевое. При выполнении программы моделирования значение переменной t всегда будет соответствовать моменту времени t_i очередного изменения состояния системы. Как только в программе выполнено моделирование состояния S_{i+1}, в которое она переходит в момент t_i, и определен момент t_{i+1} следующего изменения состояния системы, так переменной t присваивается значение t_{i+1}. Таким образом, каждое изменение значения переменной t фиксирует этап или цикл имитации процесса функционирования системы.

По окончании работы программы моделирования значение t равно моменту времени, до которого выполнена имитация работы СМО. При этом не имеет значения, по какому критерию прекращен процесс имитации.

Начальное состояние

Задание начального состояния системы S_0 перед началом моделирование вполне естественно. В качестве начального состояния может быть выбрано любое возможное состояние системы. Часто S_0 выбирается такое, при котором в системе нет ни одной заявки и все приборы свободны. При проведении серии экспериментов на модели можно за начальное состояние при каждом эксперименте принимать состояние системы, на котором закончился процесс имитации в предыдущем эксперименте. В этом случае серию экспериментов можно рассматривать как один продолжительный эксперимент.

Вектор событий

В каком бы состоянии ни находилась моделируемая система для того чтобы выяснить, каково будет ее следующее состояние, следует знать, какие локальны процессы происходят в системе и когда они закончатся. Количество происходящих локальных процессов при любом состоянии системы не превышает количества последовательностей локальных процессов, определенных для системы.

Введем в качестве элементов модели константу целого типа K со значением, равным количеству последовательностей локальных процессов или потоков основных событий, и вектор $T = [\tau_1,\tau_2,\ldots,\tau_K]$, который назовем *вектором событий*.

Если в процесс выполнения программы моделирования в некоторый момент модельного времени t только начался или уже происходит локальный процесс в i-й последовательности, то переменной τ_i будем присваивать значение момента окончания этого процесса, т. е. момента наступления ближайшего события в i-м потоке основных событий. Если же такого локального процесса нет, то этой переменной τ_i будем присваивать некоторое большое значение, которое обозначим BigC. Неважно, какое оно будет, важно, чтобы оно было заведомо больше интервала моделирования. В программах моделирования его можно принимать равным максимальному значению переменных того типа, которым описана переменная t.

Иначе значения компонент вектора событий можно определять так. Переменной τ_i будем присваивать значение момента наступления первого после t события i-го потока основных событий, если событие этого потока в принципе может изменить текущее состояние системы. Если событие i-го потока не может изменить текущее состояние, поскольку заведомо ясно, что оно наступит позже события в каком-то

другом потоке основных событий, то этой переменной будем присваивать BigC.

Перед началом моделирования именно соответствующим компонентам вектора событий следует присвоить смоделированные или полученные иным путем значения моментов окончания локальных процессов, которые происходят в системе при состоянии ее S_0. Остальным компонентам следует присвоить значение BigC. Таким образом, перед началом моделирования вместе с вектором состояния S_0 необходимо сформировать начальный вектор событий T_0.

Индекс события

Введем переменную m_s, как еще один элемент модели, и назовем её индексом очередного события

Этой переменной будем присваивать индекс компоненты текущего вектора событий, которая имеет минимальное значение. Таким образом, переменная m_s всегда будет иметь значение номера того потока, событие которого изменит текущее состояние системы. Естественно, значение m_s в программе моделирования будет определять в одном блоке с операторами нахождения минимального из значений τ_i.

Наступление первого события

При формировании начального вектора событий T_0 целесообразно одной из переменных τ_i присвоить значение, равное начальному значению модельного времени t, т. е. предположить, что один из локальных процессов, происходящий при состоянии S_0, заканчивается в момент начала моделирования. Другими словами, целесообразно предполагать, что процесс имитации функционирования СМО начинается с момента времени, когда в системе происходит некоторое событие, изменяющее ее состояние.

Это предположение не является существенным для достижения целей моделирования, но позволяет несколько упростить структуру программы моделирования и организацию длительных экспериментов. Например, если некоторый эксперимент на модели требует много часов непрерывной работы компьютера, на котором выполняется программа имитационного моделирования, то такой эксперимент можно провести в несколько этапов. После определенного времени работы модели запоминается текущее значение модельного времени t, когда происходит событие из некоторого потока, номер этого потока m_s, состояние S_n, в котором система пребывает в этот момент, моменты наступления основных событий τ_i, $i = 1, 2, ..., K$, другие данные моделирования и работа программы прерывается. В другое время, когда появляется

возможность продолжить эксперимент, сохраненные данные вводятся в программу и прерванный эксперимент продолжается.

Цикл имитации

После того, как определены все перечисленные выше исходные данные, процесс моделирования состоит в повторении следующих действий.

Прежде всего формируется новое состояние, в которое переходит система в текущий момент модельного времени t, вследствие наступившего основного события в потоке m_s, и в коррекции вектора событий.

Действия по формированию нового состояния по известному текущему следует начинать с вычисления новых значений тех переменных состояния, которые, безусловно, изменяются при наступлении данного события. Затем необходимо проверять, какие второстепенные события наступают вместе с данным основным при текущем состоянии системы, и сделать соответствующие изменения переменных состояния. При этом необходимо рассмотреть все, какие только возможны, варианты наступления второстепенных событий.

Допустим, мы в программе сформировали новое состояние системы, т. е. изменили соответствующим образом значения переменных состояния и, следовательно, определили новый вектор состояния. Теперь вновь надо определять момент наступления первого после t основного события, которое изменит это состояние. Для этого надо знать моменты наступления первых после t основных событий, т. е. надо сформировать новый вектор событий, соответствующий новому состоянию.

При формировании нового вектора событий необходимо учесть следующее.

Во-первых, часть компонент нового вектора будет равна соответствующим компонентам предыдущего. Это моменты окончания тех локальных процессов, которые происходили при предыдущем состоянии и продолжаются без изменения при новом состоянии.

Во-вторых, необходимо определить новое значение компоненты с номером m_s. Значение этой компоненты совпадает с текущим значением модельного времени. В этот момент закончился локальный процесс в последовательности m_s. Возможны два варианта. В момент t начнется новый локальный процесс в последовательности m_s и факт его окончания в принципе может изменить новое состояние. Следовательно, в этом случае необходимо смоделировать длительность

этого локального процесса (обозначим ее z) и компоненте с индексом m_s присвоить значение $t + z$. Другой вариант состоит в том, что новый локальный процесс в последовательности m_s не начнется, и тогда этой компоненте вектора событий следует присвоить значение BigC.

Например, в момент t наступило событие в потоке m_s, т. е. закончился локальный процесс в последовательности m_s, и пусть это процесс обслуживание заявки в некотором приборе. Если после этого момента прибор не принимает на обслуживание следующую заявку и переходит в состояние ожидания, то соответствующей компоненте с индексом m_s вектора событий следует присвоить значение BigC. Действительно, следующее окончание обслуживания произойдет только после того, как оно начнется, а начнется оно обязательно позже, в момент наступления какого-то другого основного события. Этим основным событием может быть приход заявки, переключение режима работы прибора или что-то другое. Если же после окончания обслуживания в приборе сразу начинается следующее, то для вычисления нового значения компоненты вектора событий с номером m_s необходимо имитировать длительность z нового обслуживания, прибавить к t и результат присвоить компоненте.

В-третьих, при формировании нового вектора событий необходимо смоделировать моменты наступления тех основных событий, которые не могли изменить прежнее состояние, но могут изменить новое, т. е. надо вычислить значения тех компонент нового вектора событий, которые в прежнем были равны BigC.

Например, до момента t прибор был свободен. Момент окончания обслуживания в нем был не определен, т. е. значение соответствующей компоненты было равно BigC. В момент t пришла заявка и началось её обслуживание в приборе. Следовательно, необходимо смоделировать длительность обслуживания z этой заявки и присвоить компоненте новое значение, равное $t + z$.

В-четвертых, в принципе возможна ситуация, когда наступление события приводит к необходимости перепланировать наступление событий других потоков. Например, в момент поступления заявки прибор обслуживал заявку и был смоделирован момент окончания этого обслуживания, т. е. было вычислено значение соответствующей компоненты, не равное BigC. По дисциплине обслуживания приход новой заявки прерывает ранее начатое обслуживание и начинает обслуживание поступившей заявки. Следовательно, надо имитировать длительность нового обслуживания z и компоненте присвоить новое значение $t + z$.

После формирования новых векторов состояния и событий для продолжения имитации находимо найти момент следующего изменения системы. Этот момент равен минимальной компоненте вектора событий. Следовательно, надо найти минимальную компоненту вектора событий. Обозначим ее значение t', а номер или индекс этой компоненты m_s'.

Вычисление t' и m_s' фактически завершает очередной этап моделирования, т. к. становится известно, что с момента *t* до момента t' система пребывает в сформированном на этапе моделирования состоянии. Также становится известно, в каком потоке основных событий произойдет следующее событие, которое изменит состояние системы. Эти данные можно использовать для вычисления оценок тех характеристик, которые необходимы для достижения целей моделирования. Далее необходимо решить, продолжать моделирование или нет. Если моделирование следует продолжить, то необходимо модельному времени *t* присваиваем значение найденной минимальной компонента t', переменной m_s присвоить значение m_s' и выполнить следующий цикл моделирования.

Итак, при каждом выполнении цикла получаем новый вектор состояния, новый вектор событий и новое значение модельного времени, т. е. моделируем процесс функционирования системы на интервале от прежнего значения модельного времени до нового значения. Эту информацию можем предварительно обработать, если необходимо, и сохранить для того, чтобы при завершении моделирования выполнить окончательную обработку в соответствии с целями моделирования.

Структура программы моделирования

Как уже отмечалось, одна из особенностей структуры, созданной по рассматриваемой технологии, состоит в разделении программы на две части: *имитационную модель* и *программу эксперимента*. В общем случае каждую часть можно представить как совокупность отдельных процедур и функций, но это могут быть и совокупности операторов в одной общей программе моделирования и крупные программные модули в отдельных файлах, допускающие автономную трансляцию. Все зависит от сложности модели и целей моделирования.

Рассмотрим одну из возможных структур программы моделирования. Блок-схема главной функции программы, с которой

начинается и которой заканчивается выполнение всей программы в каждом эксперименте, изображена на рис. 2.8.

Выполнение программы начинается с ввода всех необходимых параметров моделируемой СМО и исходных данных для проведения конкретного эксперимента. Для этого определим две функции: «Ввод параметров» и «Планирование эксперимента». Обе эти функции следует отнести к программе эксперимента. Их возможно необходимо будет создавать или существенно изменять для каждого нового эксперимента.

Рис. 2.8. Блок-схема главной функции программы моделирования

Ввод параметров

Функция ввода параметров должна выполнять следующие действия.

Во-первых, ввод параметров структуры системы, т. е. количество входных потоков, количество каналов обслуживания и обслуживающих приборов в каждом канале, количество очередей и т. д., а также количество K потоков основных событий. При необходимости в этой процедуре необходимо выполнить выделение памяти под динамические массивы, в частности под вектор событий.

Во-вторых, ввод параметров дисциплины обслуживания. К ним относятся, например, ограничения на количество заявок в системе и длины очередей, приоритеты очередей, распределения вероятностей, определяющие маршруты продвижения заявок по каналам обслуживания и т. д. Эти параметры СМО в общем случае не постоянны и могут изменяться в зависимости от целей моделирования.

В-третьих, ввод данных, которые необходимы для вычисления длительностей, происходящих в системе локальных процессов всех типов. Это параметры законов, определяющих длительности интервалов между моментами поступления заявок в каждом потоке, длительности обслуживаний заявок в приборах, количества, типы и параметры поступающих заявок и т. д.

Планирование эксперимента

В этой функции должны быть определены все исходные данные для проведения конкретного эксперимента. При этом обязательно должны быть определены следующие величины.

1. Исходные значения для датчиков псевдослучайных чисел, определяющие конкретную реализацию процесса функционирования системы. Подробнее датчики будет описаны позже, а пока будем считать, что это некоторые функции в программе, которые используются для моделирования случайных величин с заданными законами распределения.

2. Значения параметров критерия окончания эксперимента. Например, если в эксперименте необходимо смоделировать процесс функционирования СМО на интервале времени определенной длины, то должно быть введено значение момента времени окончания эксперимента. Как только в процессе моделирования значение модельного времени t превысит это значение, эксперимент следует прекратить.

3. Начальные значения переменных, которые будут использоваться для вычисления оценок характеристик моделируемой системы.

4. Начальное значение модельного времени t.

5. Значения всех переменных, которые определяют начальное состояние моделируемой системы, т. е. вектор состояния системы перед началом моделирования.

6. Значения компонент вектора событий τ_i, $i = 1, 2, ..., k,$ которые, как уже упоминалось выше, определяют моменты наступления основных событий при начальном состоянии системы. Если событие i-го потока может изменить начальное состояние системы, то должна быть задана или смоделирована длительность этого локального процесса, вычислен момент окончания процесса и значение момента присвоено переменной τ_i. Если не происходит локального процесса в i-й последовательности, то переменной τ_i должно быть присвоено значение *BigC*.

7. Рассматриваемая структура программы моделирования предполагает, что значение одной из переменных τ_i принимается равным значению t, т. е. предполагается, что первое событие, изменяющее начальное состояние системы происходит в момент начала моделирования. Следовательно, в этом случае необходимо переменной τ_i присвоить значение t, а значение ее индекса – переменной m_s.

Основной цикл моделирования

После ввода всех необходимых параметров системы и данных для эксперимента в программе выполняется основной цикл моделирования. При каждом выполнении действий основного цикла, кроме всего прочего, известно: значение модельного времени t; текущее состояние

системы и вектор событий для этого состояния; значение m_s номера потока основных событий, в котором произошло событие в момент t.

В цикле сначала выполняются действия по формированию нового состояния, в которое система переходит при наступлении события в потоке m_s в момент времени t, и корректируются значения компонент вектора событий. В предлагаемой структуре программы эти действия выполняются в блоке операторов «Модель», из которой вызываются *функции обработки событий*. Каждому потоку основных событий соответствует одна функция. В зависимости от значения m_s выполняется обращение к одной из них. Каждая из этих функций выполняет все действия по формированию нового состояния системы и нового вектора событий в каком бы текущем состоянии не находилась система. В свою очередь, при выполнении любой из этих функций могут вызываться функции обработки второстепенных событий, некоторые другие вспомогательные функции, также входящие в состав имитационной модели. Обязательно из этих функций должны быть обращения к *функциям моделирования локальных процессов*, которые следует отнести к программе эксперимента.

После того, как в основном цикле моделирования в одной из функций обработки событий сформировано новое состояние системы и соответствующим образом изменен вектор событий, находится момент t' наступления следующего события и номер потока m'_s, в котором это событие произошло. Для этого находится минимальная компонента вектора событий и ее номер. Эти значения и присваиваются переменным t' и m'_s.

Далее из основного цикла моделирования происходит обращение к функции «Обработка результатов». Эта функция должна возвратить значение, по которому принимается решение, закончить основной цикл моделирования и программу моделирования в целом или эксперимент необходимо продолжить. Если необходимо продолжить, то следует переменным t и m_s присвоить значения t' и m'_s и вернуться к началу основного цикла моделирования.

Обработка результатов

Эту функцию можно считать основной функцией программы эксперимента, как части всей программы моделирования. Обращение к этой функции происходит из основного цикла моделирования в тот момент, когда сформировано очередное состояние СМО, скорректирован вектор событий, определен момент t', до которого

система пробудет в сформированном состоянии, и известен тип события, которое произойдет в этот момент. Другими словами, обращение к этой функции происходит после того, как выполнено моделирование системы на интервале от t до t', когда выполнен этап моделирования. Следовательно, первое, что необходимо выполнять в функции, так это обрабатывать, анализировать, накапливать и сохранять информацию об этапах моделирования в соответствии с целями моделирования.

Так, например, для того, чтобы вычислить среднюю длину некоторой очереди, необходимо длину этой очереди при текущем состоянии системы умножить на длительность пребывания системы в этом состоянии, равную разности $t' - t$, и сложить с аналогичными произведениями, полученными на предыдущих этапах моделирования. Естественно, для реализации таких вычислений при обращении к функции «Обработка результатов» необходимо определить некоторую переменную с нулевым начальным значением и использовать ее для накопления суммы произведений. Определить такую переменную следует в функции «Планирование эксперимента». При завершении эксперимента необходимо накопленную сумму произведений поделить на разность значений модельного времени t в конце и начале эксперимента, т. е. на длительность эксперимента, и в результате получить среднюю длину очереди в данном эксперименте.

Другой пример. Необходимо в эксперименте оценить вероятность того, что некоторый прибор окажется занятым в произвольный момент времени. В этом случае также при планировании эксперимента необходимо определить некоторую переменную с нулевым начальным значением. Затем при каждом обращении в функции «Обработка результатов» необходимо проверять условие занятости этого прибора. Если при очередном состоянии моделируемой системы прибор оказывается занятым, то к прежнему значению этой переменной надо добавить длительность пребывания системы в текущем состоянии. Как и в предыдущем случае при завершении эксперимента надо накопленную сумму поделить на значение модельного времени. В результате получится искомая оценка вероятности занятости прибора.

И, наконец, при отладке модели можно не проводить текущую обработку результатов, а просто выводить на экран или в файл информацию о состояниях системы на интервале моделирования. Например, значения t, m_s и всех переменных состояния систем. При моделировании на диаграмме такая информация о состояниях системы приведена в таблицах. После эксперимента по этим значениям можно

проверить, действительно ли имитационная модель правильно моделирует переходы системы из состояния в состояние.

Второе действие, которое выполняется в функции «Обработка результатов», состоит в проверке критерия окончания эксперимента. Критерий может быть любой. Эксперимент можно продолжать, например, до тех пор, пока модельное время не достигнет определенной величины, или пока система не перейдет некоторое определенное количество раз из состояния в состояние, или, наконец, пока оценка некоторого параметра системы не будет вычислена с заданной точностью. Естественно, при планировании эксперимента необходимо задать параметры критерия.

При выполнении критерия окончания эксперимента на очередном этапе моделирования в функции «Обработка результатов» должны быть выполнены действия по окончательной обработке результатов эксперимента и выдачи их пользователю. После этого функция должна завершить работу и в главную процедуру программы моделирования должно быть передано, например, нулевое значение, при котором выполняется условие окончания основного цикла моделирования и завершается работа всей программы моделирования.

Если критерий окончания эксперимента не выполняется, то выполнение функции «Обработка результатов» завершается и в главную функцию программы моделирования передается значение, при котором в основном цикле моделирования выполняется условие продолжения моделирования.

Предложенная здесь структура программ имитационного моделирования систем массового обслуживания достаточно проста в реализации на универсальных языках программирования. Это очевидно из следующего примера и из примеров, приведенных в последней главе.

Функциям моделирования локальных процессов

В общем случае таких функций должно быть столько, сколько в системе массового обслуживания задано различных законов, определяющих длительности локальных процессов. При обращении к каждой из этих функций она должна возвращать длительность соответствующего локального процесса.

Целесообразно функции моделирования локальных процессов создавать таким образом, чтобы тело функции можно было изменять при проведении различных экспериментов, а обращения к этим функциям оставались неизменными. Например, в качестве аргумента функции использовать указатель на переменную, массив или структуру,

где хранятся параметры закона, определяющего длительности соответствующего локального процесса. Значения параметров, как уже отмечалось, должны быть заданы в функции «Ввод параметров». Это позволит в экспериментах менять законы, определяющие длительности локальных процессов и соответственно тела функций, но не изменять основные функции модели.

В СМО чаще всего предполагается, что длительности локальных процессов случайные величины с заданными законами распределения. Некоторые алгоритмы моделирования случайных величин приведены в следующей главе.

Функции обработки событий

Функции обработки событий являются главными функциями имитационной модели как составной части программы моделирования. Именно эти функции составляют основу имитационной модели.

Каждому потоку основных событий или каждой последовательности локальных процессов соответствует одна функция обработки событий. Однако каждая функция может соответствовать нескольким последовательностям, если эти последовательности образуют однотипные локальные процессы. Например, если некоторый элемент моделируемой реальной системы может одновременно взаимодействовать с несколькими дискретными объектами, то в создаваемой СМО обычно этот элемент моделируется некоторым количеством совершенно одинаковых приборов. Каждый из этих приборов в каждый момент времени может обслуживать только одну заявку. Все локальные процессы обслуживания в этих приборах однотипные, но образуют столько последовательностей, сколько определено приборов. Следовательно, для обработки любого события окончания обслуживания в любом приборе можно использовать одну функцию обработки событий.

Обращения к функциям обработки событий происходит в начале основного цикла моделирования, но вызывается каждый раз только одна функция. В моменте обращения к любой из функций известно значение модельного времени t, индекс m_s наступившего в момент t события, вектор событий и вектор состояния системы. При первом выполнении основного цикла программы эти данные формируются в функции «Планирование эксперимента». При последующих выполнениях эти данные формируются на предыдущем этапе выполнения цикла. Другими словами при каждом обращении к одной из этих функций известно состояние системы, какие локальные процессы происходят при

этом состоянии, какой из этих процессов закончился в текущий момент модельного времени t. А если последовательность локальных процессов с индексом (номером) m_s образуют процессы разных типов, то по состоянию системы также должно быть известно, локальный процесс какого типа закончился.

Задача, которую решает каждая функция обработки событий, состоит в формировании нового вектора состояния и корректировки вектора событий при наступлении события в потоке, которому функция соответствует. При этом естественно учитывается текущее состояние, какое бы оно не было, и тип завершившегося локального процесса.

Видимо, нельзя предложить некий универсальный алгоритм или правило написания функций обработки событий, но можно дать следующие рекомендации.

Во-первых, при написании текста функции обработки любого основного события необходимо описать действия по изменению состояния системы и корректировки вектора состояния, которые безусловно произойдут при любом состоянии системы.

Например, при создании функции, которая будет обрабатывать факт поступления в систему извне новой заявки, надо предусмотреть оператор вычисления момента поступления следующей заявки, т. е. надо скорректировать вектор событий, изменить соответствующий компонент τ_i вектора событий. Необходимо обратиться к соответствующей функции моделирования локального процесса, полученное значение прибавить к текущему значению t и результат присвоить τ_i. Если по дисциплине обслуживания каждая поступающая заявка принимается в систему, а в векторе состояния системы определена переменная со значением общего количества заявок в системе, то в функции безусловно должен быть оператор увеличения на единицу значения этой переменной.

Во-вторых, Если функция обработки событий соответствует последовательности, состоящей из локальных процессов разных типов, то необходимо предусмотреть в функции операторы определения, какого типа процесс закончился. В соответствии с этим должна быть операторы формирования состояния и корректировки вектора событий при окончании процесса каждого типа.

В-третьих, если функция обрабатывает события окончания обслуживаний заявок в приборе, то целесообразно сначала включить в функцию операторы, которые формируют состояние, вызванное уходом заявки, а затем операторы формирования состояния обслужившего заявку прибора.

В четвертых. Не следует, например, сначала включить в функцию операторы формирования состояния, а затем операторы корректировки вектора событий. Если формируется состояние, при котором начинается новый локальный процесс, то сразу необходимо вычислить момент окончания этого процесса, т. е. скорректировать вектор событий.

И, наконец, функции обработки событий надо писать так, как будто нВ языке программирования описываете дисциплину обслуживания с точки зрения происходящих в системе событий.

Рассмотрим далее один пример создания программы моделирования. Еще несколько примеров приведены в последней главе.

Пример программа моделирования

Возьмем снова систему массового обслуживания «Оптовый магазин», которая описана в первой главе. В этой главе для неё выбран и описан один из вариантов вектора состояния, определен вектор событий и выполнена имитация работы системы без программирования.

Напишем теперь на языке «С» главную функцию программы моделирования СМО, блок-схема которой представлена на рис 2.8, функции формирования состояний и приведем описание всех необходимых переменных. Будем использовать только простейшие средства языка, что позволит легче разобраться в сути функций, понять смысл выполняемых в них действий и при необходимости переписать их на любом другом языке программирования. Также для упрощения текстов функций программы все параметры СМО, переменные модели, вектора состояний и событий определим как глобальные константы, переменные и массивы. В частности опишем следующие константы.

```
#define BigC 1E+300
#define N 100       //максимальное количество заявок в СМО
```

Для ограничения на количество одновременно находящихся в системе заявок как параметра СМО надо бы ввести переменную *N* и при каждом выполнении программы в функции «Ввод параметров» присваивать ей некоторое конкретное значение. Однако для упрощения программы этот параметр зададим как константу, т. е. будем предполагать, что покупатель, застав 100 уже находящихся в магазине покупателей, не заходит в магазин.

Переменные модели опишем следующим образом

```
double t;       //модельное время t
```

```
double tnew;        //момент наступления
```
следующего события t'

```
int ms;             //индекс текущего события m_s
int msnew;      //индекс следующего события m_s'
```

Ранее для СМО были определены 4 последовательности локальных процессов. Процессы ожидания поступления в систему заявок образуют одну последовательность локальных процессов. Процессы обслуживания в приборах образуют еще три последовательности, причем в первых двух приборах эти последовательности образуют процессы двух типов. В соответствии с этим вектор событий определим так:

```
double tau[K];      //вектор событий: tau[0] –
```
момент поступления

```
                    //заявки; tau[1], tau[2] и tau[3]
              – моменты
                    //окончаний обслуживаний в
              приборах 1, 2 и 3.
```

Будем предполагать, что функции «Ввод параметров», «Планирование эксперимента» и «Обработка результатов», которые представлены на блок-схеме главной функции программы моделирования, описаны следующим образом:

```
void param(void);
void planexp(void);
int exper(void);
```

Тексты этих функций здесь приводить не будем, поскольку написание этих функций не представляет особого труда

В СМО предусмотрены 6 законов, определяющих длительности 6 типов локальных процессов. Следовательно, в программе моделирования необходимо определить 6 функций вычисления длительностей этих процессов. Однако вполне логично предположить, что все обслуживания первичных заявок в обоих приборах определяются одним законом, но с разными параметрами, которые определяют интенсивности работы приборов. Так можно учесть разные интенсивности работы продавцов при одинаковых правилах обслуживания покупателей. Аналогично можно считать, что и законы

вторичного обслуживания заявок одинаковы, но имеют различные параметры. С учетом этих предположений опишем в программе следующие функции для моделирования длительностей локальных процессов.

```
double fvx(void)    //длительность ожидание заявки
double fo1(int);    //длительности первичного обслуживания
double fo2(int);    //длительности вторичного обслуживания
double f3(void)     //дительности обслуживания в третьем приборе
```

В функции `fo1` и `fo2` перед обращением должен быть передан индекс прибора, длительность обслуживания в котором необходимо вычислить. Индекс первого прибора пусть равен нулю, второго – единица.

Определять все эти функции и их параметры следует при постановке конкретных экспериментов. Параметры законов обслуживания естественно должны быть описаны и определены в функции «Планирования эксперимента» в программе моделирования.

Для описания состояния СМО будем использовать определенные ранее следующие массивы и переменные.

```
int k1[2];//количество заявок в первичных очередях
int k2[2];//количество заявок во вторичных очередях
int s[2]; //состояния первого и второго приборов
```

Первому прибору соответствует первый элемент массивов с индексом 0, второму – с индексом 1.

```
int k3;         //количество заявок в очереди к третьему прибору.
int r[N]; //состояние очереди к третьему прибору
int k0;         //общего количества заявок в системе
```

В СМО определены 4 потока основных событий. Поступая формально надо бы создать 4 функции формирования состояний

системы. Каждая функция при этом формировала бы состояние системы и корректировала вектор событий при наступлении событий в одном потоке. Например, функция sob0 формировала бы состояние системы после прихода очередной заявки, а функции sob1, Sob2 и sob3 – после окончаний обслуживаний в приборах. Но поскольку два первых прибора работают совершенно одинаково, то целесообразно вместо функций sob1 и Sob2 создать одну sob12. Поскольку переменные модели договорились описывать как глобальные, то эти функции можно описать так:

```
   void sob0(void);   //поступление заявки
   void sob12(void);  //окончание обслуживания в
приборах 1 и 2
   void sob3(void);   //окончание обслуживания в
приборе 3
```

С учетом введенных переменных и описаний функций главная функция программы моделирования может быть такой.

```
   void main(void)
     { int i;     //рабочая переменная
     param();     //ввод параметров
     planexp();       //планирование эксперимента
   C:                 //начало цикла моделирования
     //обрацения к функциям обработки событий
     if (ms==0)       sob0();
     else if (ms<=3)  sob12();
          else        sob3();
     //определение момента наступления и индекса
следующего событияи
     tnew = BigC;
     for (i=0;i<K;i++)
       if (tau[i]<tnew) {tnew=tau[i]; msnew=i;}
     i=exper();       //обращение к функции
Эксперимент
     //проверяем условие окончания эксперимента
```

```
    if (i>0) {        //если эксперимент необходимо
продолжить, то
       t=tnew; ms=msnew;//времени и индексу события
новые значения и
       goto C;}       //к началу цикла моделирования
    }   //конец программы
```

Опишем теперь достаточно подробно функции формирования состояний, как основные функции программы моделирования.

Функция sob0

Эта функция выполняется в том случае, когда закончится локальный процесс ожидания заявки. Независимо от того, в какое состояние перейдет система после этого события, в системе начнется новый локальный процесс ожидания следующей заявки. Это значит необходимо смоделировать момент прихода следующей заявки и вычислить новое значение оператором

```
tau[0]=t+fvx();
```

Поступление новой заявки не изменяет состояние системы, если в ней уже есть *N* заявок. Выполнение функции необходимо продолжить только в случае, когда поступившая заявка принимается в систему. Следовательно, в функции должен быть оператор

```
if (k0<N)
```

После проверки условия приема заявки в систему в функции обязательно должен быть оператор

```
k0++;
```

фиксирующий факт добавления поступившей заявки к общему количеству заявок в системе.

Согласно дисциплине обслуживания принятая в систему заявка должна быть направлена в первый или во второй прибор в зависимости от длин первичных очередей к этим приборам. Другими словами, следствием принятия заявки в систему является наступление одного из второстепенных событий: увеличение длины первичной очереди к первому или ко второму прибору.

В программе длинам первичных очередей соответствуют значения переменных `k1[0]` и `k1[1]`. Следовательно, необходимо сравнить эти значения и зафиксировать факт постановки заявки в первичную очередь, увеличив на единицу значение соответствующе переменной. Это можно сделать следующими операторами:

```
if (k1[0]<=k1[1]) i=0; else i=1;
k1[i]++;.
```

Перед этим в функции должна быть описана целочисленная рабочая переменная i, которая в данном случае получает значение индекса того прибора, в который будет направлена принятая в систему заявка.

Постановка заявки в одну из первичных очередей может привести к наступлению еще одного второстепенного события – начало обслуживания этой заявки в приборе. Это произойдет, если текущее состояние таково, что соответствующий прибор свободен. По дисциплине обслуживания прибор должен начать обслуживать эту заявку, но до окончания обслуживания заявка должна оставаться в очереди. В функции эта ситуация может быть описана следующими операторами:

```
if (s[i]==0) {s[i]=1;  tau[i]=t+fo1(i);}
```

В этих операторах проверяется, свободен ли прибор, и если свободен, то изменяется его состояние и вычисляется момент окончания обслуживания заявки.

Очевидно, что других второстепенных событий кроме перечисленных при поступлении заявки не произойдет в каком бы текущем состоянии не находилась система. Таким образом, в окончательном варианте функцию можно записать в виде

```
void sob0(void)          //поступила заявка на вход
  {int i;              //рабочая переменная
   tau[0]=t+fvx();       //моделируем приход следующей заявки
   if (k0<N)             //если заявка в СМО принимается,
     {k0++;            //увеличиваем количество заявок
        //определяем индекс прибара, к которому направить заявку
      if (k1[0]<=k1[1]) i=0; else i=1;
      k1[i]++;           //увеличиваем первичную очередь
```

```
      if (s[i]==0)              //если прибор
свободен, то
        {s[i]=1;          //заявку начинаем
обслуживать и
         tau[i+1]=t+fo1(i);}   //моделируем
окончание обслуживания
      }
    }                  //конец sob0
```

Функция sob12

Эта функция выполняется, когда наступившем основным событием является окончание обслуживания заявки в первом или во втором приборе. При этом значение индекса события `ms` равно 1 или 2. Введем целочисленную переменную `n` и присвоим ей значение `ms-1`, равное индексу того прибора, в котором закончилось обслуживание заявки.

Первое, что необходимо сделать для формирования состояния системы в этом случае, так это выяснить, первичное или вторичное обслуживание закончилось. Если вторичное, то обслуженная заявка покидает систему и, следовательно, необходимо уменьшить на единицу общее количество заявок в системе и длину вторичной очереди к данному прибору. В противном случае следует отразить изменения состояния, связанные с поступлением обслуженной заявки в очередь к третьему прибору.

При поступлении заявки в очередь к третьему прибору необходимо увеличить длину очереди `k3` на единицу и запомнить номер прибора, в котором заявка прошла первичное обслуживание. Для этого в состав переменных состояния системы включен массив *r*. Если в очереди к третьему прибору стоят `k3` заявок, то переменные `r[0], r[1], ..., r[k3-1]` имеют уже значения номеров тех приборов, в которых эти заявки прошли первичное обслуживание. Следовательно, номер прибора, в котором обслужилась поступающая в очередь заявка, должен быть присвоен переменной `r[k3]`. После этого значение `k3` можно увеличить на 1. Таким образом, в процедуре действия по формированию состояния системы при постановке заявки в очередь к третьему прибору можно выполнить одним оператором

```
r[k3++]=ms;
```

Поступление заявки в очередь к третьему прибору может привести к наступлению еще одного вспомогательного события. Если заявок в

очереди не было, то начнется обслуживание поступившей заявки. Следовательно, в этом случае необходимо смоделировать длительность ее обслуживания и переменной tau[3] присвоить значение момента окончания этого обслуживания.

Закончив формирование состояния системы, вызванные выходом заявки из обслужившего ее прибора, теперь необходимо сформировать состояние самого прибора. Для этого необходимо проверить состояние очередей к прибору. Если есть заявки во вторичной очереди, то прибор должен взять на обслуживание заявку из этой очереди. При этом необходимо присвоить значение 2 переменной, определяющей состояние прибора, смоделировать длительность обслуживания и вычислить значение переменной tau[ms]. Если эта очередь пуста, то необходимо проверить другую очередь. Если в ней есть ли заявки, ожидающие первичное обслуживание, то необходимо принять на обслуживание заявку из этой очереди. В этом случае переменной, определяющей состояние прибора, следует присвоить значение 1 и вычислить значение переменной tau[ms]. Если обе очереди пусты, то прибор должен перейти в состояние ожидания, переменная состояния прибора должна получить нулевое значение. Поскольку в данном случае локальный процесс не начинается, то переменной tau[ms] необходимо присвоить значение BigC. Таким образом, функцию можно записать в следующем виде:

```
   void sob12(void)    //закончилось обслуживание в 1 или 2 приборе
     { int n=ms-1;     //индекс прибора, обслужившего заявку
     if (s[n]==2)      //если закончилось вторичное обслуживание,
       {k0--; k2[n]--;}   //заявка уходит из очереди и из системы
     else              //если закончилось первичное обслуживание,
       { k1[n]--;          //заявка покидает первичную очередь и
       r[k3++]=ms;         //становится в очередь к прибору 3
```

```
      if (k3==1)     //если прибор свободен,
начинаем обслуживание и
         tau[3]=t-f3();}//моделируем окончание
обслуживания
         //формируем состояние прибора после ухода
заявки
      if (k2[n]>0)   //если вторичная очередь не
пуста,
         { s[n]=2;    //начинаем обслуживание заявки
из очереди
         tau[ms]=t-fo2(n); }  //моделируем
окончание обслуживания
      else                    //если вторичная
очередь пуста, но
         if (k1[n]>0) //в первичной есть заявки, то
           { s[n]=1;  //начинаем обслуживание заявки
очереди
           tau[ms]=t-fo1(n); }//моделируем
окончание обслуж.
         else     //если обе очереди к прибору
пусты, то
           { s[n] = 0;     //прибор переходит в
состояние ожидания
           tau[ms]=BigC; } }  //окончание
обслуживания не определено
    }            //конец sob1
```

Функция sob3

Процедура sob3 обрабатывает основные события, каждое из которых есть факт окончания обслуживания заявки в третьем приборе.

По дисциплине обслуживания заявка, обслуживание которой закончилось в третьем приборе, должна вернуться на вторичное обслуживание в тот прибор, в котором она прошла первичное обслуживание. Следовательно, первыми действиями в процедуре должны быть определение прибора, в который должна быть направлена заявка на вторичное обслуживание, и постановка заявки во вторичную

очередь к этому прибору. Если прибор в этот момент свободен, то надо также направить заявку на обслуживание и вычислить момент окончания обслуживания.

Введем вспомогательную целочисленную переменную *i* и присвоим ей значение индекса прибора, в котором прошла первичное обслуживание заявка, покидающая третий прибор.

```
i=r[0]-1;
```

Тогда первыми операторами процедуры могут быть следующие:

```
k2[i]++;
if (s[i]==0){s[i]=2; tau[i+1]=t-sv(b[i]);}
```

Уход заявки из третьего прибора приводит к изменению состояния очереди к этому прибору. Поскольку состояние этой очереди определяется переменными `k3` и `r[0], r[1], ..., r[N]`, следовательно, необходимо скорректировать значения этих переменных. Это можно сделать, выполнив следующие операторы:

```
k3--;
for (i=0;i<k3;i++) r[i]=r[i+1]; r[i]=0;
```

Первый оператор уменьшает на единицу количество заявок в очереди к третьему прибору, а остальные «сдвигают» очередь и освобождают последнее занятое место в этой очереди. После выполнения этих операторов состояние третьего прибора будет определяться новым значением переменной `r[0]`. Если `r[0]` отлично от нуля, т. е. в очереди есть заявка, то третий прибор должен начать ее обслуживание. Следовательно, необходимо смоделировать длительность этого обслуживания и вычислить новое значение переменной `tau[3]=t+f3()`.

Если `r[0]` равно нулю, т. е. очередь к прибору пуста, то прибор должен перейти в состояние ожидания. В этом случае переменной `tau[3]` следует присвоить значение `BigC`.

Итак, в окончательном варианте функция `sob3` запишется в виде:

```
  void sob3 (void)   //закончилось обслуживание в
3-м приборе
    { int i;     //рабочая переменная
    int n=r[0]-1;    //индекс прибора, в который
должна быть
```

```
                    //направлена обслуженная в приборе
               заявка
     k2[n]++;     //ставим  заявку во вторую очередь
к  прибору
     if (s[n]==0)     //если прибор свободен, то
начинается
       {s[n]=2;       //вторичное обслуживание
заявки и
       tau[i+1]=t-fo2(n); }    //моделируем
окончание обслуживания
     k3--;            //удаляем заявку из очереди к
3-му прибору и
     for (i=0;i<k3;i++)    //сдвигаем очередь
       r[i]=r[i+1];
     r[i]=0;
     if (k3>0)        //если еще есть заявка в этой
очереди, то
        tau[3]=t-f3();    //моделируем момент
окончания  обслуживания
     else             //очередной заявки, а если
заявок нет,
       tau[3]=BigC;   //окончание обслуживания не
определено
     }           //конец sob3
```

Итак, как видно из приведенных примеров, процедуры обработки событий по сути дела представляют собой запись на языке программирования дисциплины обслуживания моделируемой СМО с точки зрения происходящих в ней событий. Каждому типу основного события соответствует процедура, в которой сначала выполняются те действия по формированию векторов состояния и событий, которые обязательно происходят при наступлении данного события. Затем анализируются все возможные состояния системы, при которых может произойти данное событие и в зависимости от того, которое из них является текущим в данный момент, выполняются соответствующие

изменения состояния и корректируются моменты наступления основных событий, т. е. изменяется вектор событий.

Глава 3.

Моделирование случайных величин

Стандартная случайная величина

Имитационное моделирование предполагает моделирование случайных событий, дискретных и непрерывных случайных величин с заданными законами распределения, потоков случайных событий, случайных процессов и т. д. Как известно [2], для того чтобы вычислять значения любых случайной величин, достаточно уметь находить значения какой-нибудь одной случайной величины с определенным законом распределения, ибо всегда можно подобрать такую функцию от этой случайной величины, которая позволит преобразовать это значение в значение случайной величины с требуемым законом распределения. Поэтому мы сначала рассмотрим вопрос о том, как получать на компьютере значения одной, «стандартной» или «базовой», случайной величины.

Случайные числа и случайные цифры

Как правило, в качестве стандартной выбирают непрерывную случайную величину γ, равномерно распределенную в интервале (0,1). Напомним основные характеристики этой величины. При $0 < x < 1$ плотность $p_\gamma(x) = 1$, а функция распределения $F_\gamma(x) = x$, математическое ожидание $M_\gamma = 1/2$, дисперсия $D_\gamma = 1/12$.

В качестве стандартной можно также использовать дискретную случайную величину ε, которая с одинаковой вероятностью может принимать 10 значений 0, 1, 2, . . . , 9. Мы будем называть величину γ *случайным числом*, а величину ε *случайной цифрой*. Иногда ε называют десятичной случайной цифрой, чтобы отличить ее от *двоичной случайной цифры* – случайной величины α, которая с равной вероятностью 0,5 принимает значения 0 и 1.

Чтобы установить связь между γ и ε, разложим число γ в бесконечную десятичную дробь:

$$\gamma = 0,\varepsilon_1\varepsilon_2...\varepsilon_k...$$

Последняя запись означает, что

$$\gamma = \sum_{k=1}^{\infty} \varepsilon_k 10^{-k}.$$

Оказывается, и это можно доказать, что десятичные цифры $\varepsilon_1, \varepsilon_2, ..., \varepsilon_k, ...$ случайного числа γ представляют собой независимые случайные цифры. Обратно, если $\varepsilon_1, \varepsilon_2, ..., \varepsilon_k, ...$ – независимые случайные цифры, то формула $\gamma = 0,\varepsilon_1\varepsilon_2...\varepsilon_k...$ определяет случайное число.

Приближенные случайные числа

В вычислениях всегда используют числа с конечным количеством десятичных знаков, поэтому вместо случайных чисел γ употребляют конечные десятичные дроби $\gamma' = 0,\varepsilon_1\varepsilon_2...\varepsilon_n$. Никаких специальных исследований по этому поводу проводить не будем. Будем считать, что здесь имеет место ошибка округления такая же, как при любых приближенных вычислениях. Далее, говоря о случайных числах, будем иметь в виду приближенные случайные числа, и обозначать их будем греческими буквами без штриха.

Отметим одно простое свойство чисел γ, которое теряется при таком приближении. Свойство это состоит в том, что, если случайное число γ умножить на произвольное целое положительное число g, то случайная величина $\eta = Д(g\gamma)$ равномерно распределена в интервале (0,1). Ясно, что приближенная случайная величина $\gamma' = 0,\varepsilon_1\varepsilon_2...\varepsilon_n$ не обладает таким свойством, так как при $g = p10^n$, где p – целое, $Д(g\gamma') = 0$.

Здесь и далее будем использовать следующие обозначения:

$Ц(x)$ – целая часть числа x, т. е. наибольшее целое число, не превосходящее x (так $Ц(3,6) = 3$, а вот $Ц(–3,6) = –4$);

$Д(x)$ – дробная часть числа x, т. е. $Д(x) = x – Ц(x)$, которая всегда неотрицательна .

Три способа получения случайных величин

Таблицы случайных цифр

Предположим, что мы осуществили N независимых опытов, в результате которых получили N случайных цифр $\varepsilon_1, \varepsilon_2, ..., \varepsilon_N$. Записав эти цифры (в порядке появления) в таблицу, получим то, что называется *таблицей случайных цифр*.

Способ употребления такой таблицы весьма прост. Если в ходе расчета некоторой задачи нам потребуется очередная случайная цифра, то мы можем взять очередную цифру из этой таблицы. Если нам понадобится случайное число γ, то мы можем взять из таблицы n очередных цифр $\varepsilon_{s+1}\varepsilon_{s+2}...\varepsilon_{s+n}$ и считать, что $\gamma = 0,\varepsilon_{s+1}\varepsilon_{s+2}...\varepsilon_{s+n}$. Выбирать цифры из такой таблицы в случайном порядке необязательно. Их

можно выбирать подряд. Но, конечно, можно начинать с любого места, читать в любом направлении, использовать любой заранее заданный алгоритм выбора, не зависящий от конкретных значений цифр таблицы.

Все сказанное выше относится к «идеальной» таблице случайных цифр и не вызывает никаких сомнений. Мы не будем подробно останавливаться на трудностях, возникающих при переходе к «реальным» таблицам, но отметим некоторые из них.

Во-первых, изготовление хорошей таблицы – весьма сложное дело, ибо в любом реальном опыте всегда возможны ошибки. Например, в 1955 году была опубликована таблица, содержащая миллион случайных цифр. Для её изготовления была построена и тщательно отлажена специальная «рулетка» с использованием электроники. Тем не менее, после некоторого периода хорошей работы она стала выдавать, как показала проверка, не равновероятные цифры. Таким образом, проверка «качества» таблицы абсолютно необходима. Никакие априорные соображения о тщательности постановки опыта не гарантируют нас от ошибок.

Во-вторых, отметим известный парадокс: в большой таблице всегда найдутся плохие участки. Например, в таблице, содержащей 10^{100} цифр, вполне вероятно найти 100 нулей подряд. Очевидно, самостоятельное использование таких участков недопустимо.

В настоящее время таблицы случайных цифр (или более разнообразные таблицы случайных величин) используют главным образом при расчетах вручную. Для расчетов на компьютерах они практически не пользуются. Основная причина этого – чисто техническая: оперативная память современных компьютеров сравнительно мала и большая таблица туда не помещается; а обращение к внешним запоминающим устройствам сильно замедляет счет. Вторая причина – ограниченность объема существующих таблиц, – по-видимому, играет второстепенную роль: можно заготовить таблицу любого объема, если это потребуется.

Датчики случайных чисел

Генераторами или *датчиками случайных величин* называют различные технические устройства, вырабатывающие случайные величины. Чаще всего для построения датчика используют «шумящие» радиоэлектронные приборы (диоды, тиратроны, газотроны и др.). Не вдаваясь в технические подробности, рассмотрим один из возможных способов построения датчика, вырабатывающего случайные двоичные цифры α.

Нетрудно представить себе счетчик, который подсчитывает количество флуктуаций напряжения шумящего прибора, превышающих заданный уровень, за фиксированное время Δt. Если это количество четное, то счетчик выдает 0, иначе – выдает 1. Если вероятности появления 0 и 1 в таком процессе равны между собой, то можно считать, что устройство вырабатывает случайную последовательность двоичных цифр.

Если вероятность p появления нуля отлична от половины, то можно ввести какую-нибудь схему стабилизации вероятности. Например, можно группировать цифры парами и выдавать 0 при получении пары (0 1) и 1 – при получении пары (1 0), а пары (0 0) и (1 1) просто опускать.

Обычно реальные датчики случайных чисел, которые раньше широко использовались на ЭВМ первых поколений, содержали m генераторов описанного типа, работающих независимо, и выдавали приближенное случайное число $\gamma = 0,\alpha_1\alpha_2...\alpha_m$, записанное в форме m-разрядной двоичной дроби. Для случайных чисел отводилась специальная ячейка в оперативной памяти, и скорость генерирования их была столь велика, что на каждом такте работы ЭВМ в этой ячейке получается новое случайное число.

Применение датчиков случайных чисел свободно от тех недостатков, которые препятствуют широкому применению таблиц: не требуется места во внутреннем накопителе и запас чисел неограничен. Но эти датчики имели свои недостатки, и эти недостатки оказались настолько серьезными, что сейчас генераторы случайных чисел практически не используются.

Во-первых, числа, выработанные датчиком, нельзя воспроизвести. Это затрудняет контроль расчетов и не позволяет при необходимости повторить, например, при моделировании реализацию некоторого случайного процесса в других экспериментах.

Во-вторых, приходится содержать и эксплуатировать дополнительное устройство, которое требует ухода и регулярной проверки «качества» вырабатываемых чисел с помощью специальных тестов.

Метод псевдослучайных чисел

Мы видели, что пригодность случайных чисел определяется, в конечном счете, не процессом их получения, а тем, удовлетворяют ли они некоторым тестам, которые используются для проверки их «качества». Но в таком случае совершенно безразлично, как эти числа

получены, они могут быть даже сосчитаны по какой-нибудь формуле, лишь бы они удовлетворяли тестам!

Такая точка зрения иногда вызывает возражения, ибо, вычисляя числа по формуле, мы тем самым сразу отказываемся от их «случайности». Любопытно, что такого рода возражения чаще высказывают не математики, хотя, казалось бы, математики должны более решительно протестовать против нестрогости. Дело в том, что математики лучше понимают, что слово «случайность» использовано здесь не в математическом, а скорее в обыденном смысле. С точки зрения математика, равномерно распределенная случайная величина γ – это абстрактное понятие; и только опыт может убедить нас в том, что какая-либо конкретная последовательность чисел γ_1, γ_2, ..., γ_n обладает интересующими нас свойствами независимых случайных чисел γ.

Числа γ_1, γ_2, ..., γ_n, которые вычисляются по какой-либо заданной формуле и могут быть использованы вместо случайных чисел при решении некоторых задач, называются *псевдослучайными числами*. Процедуры или функции, которые используются в программах для получения псевдослучайных чисел, обычно называют *датчиками псевдослучайных чисел*.

Следует заметить, что не существует универсальных тестов, обеспечивающих возможность использования проверенных случайных или псевдослучайных чисел в любых задачах. Каждый тест связан с каким-то достаточно широким, но ограниченным классом задач, и правильнее было бы говорить о таблицах, датчиках или псевдослучайных числах для определенных классов задач. Действительно, любой реальный объект лишь приближенно описывается математическим понятием, и всегда можно указать задачи, в которых данная математическая модель непригодна. Например, как бы аккуратно ни была начерчена прямая, при достаточно большом увеличении она превратится в волнистую или даже прерывистую линию.

Конкретная последовательность псевдослучайных чисел сходна с таблицей случайных чисел. Ее можно один раз тщательно проверить и затем многократно применять. Запас чисел в такой последовательности, как мы увидим позднее, тоже ограничен. Но метод псевдослучайных чисел свободен от главного недостатка таблиц: существуют простые формулы для расчета псевдослучайных чисел такие, что на получение каждого числа затрачивается всего около десятка команд процессора, а программа расчета занимает в памяти лишь несколько байтов.

В настоящее время из трех рассмотренных методов получения случайных чисел практически используется только метод псевдослучайных чисел. Поэтому рассмотрим его более подробно, но не станем вдаваться в теорию метода, а ограничимся изложением некоторых фактов, главным образом эмпирического характера.

Большинство алгоритмов, используемых на практике для получения псевдослучайных чисел, представляют собой рекуррентные формулы первого порядка:

$$\gamma_{n+1} = \Phi(\gamma_n),$$

где начальное число γ_0 задано.

Функции $y = \Phi(x)$ датчиков должны удовлетворять определенным требованиям. Легко показать, что функция $y = \Phi(x)$, изображенная на рис. 3.1*а*, не может породить хорошую последовательность псевдослучайных чисел $\gamma_1, \gamma_2, \ldots$ В самом деле, если по настоящим случайным числам построить точки с координатами (γ_1,γ_2), (γ_3,γ_4), . . . , то они равномерно расположатся в единичном квадрате $0 < x < 1$, $0 < y < 1$, в то время как соответствующие точки, построенные по числам $(\gamma_1, \Phi(\gamma_1))$, $(\gamma_3, \Phi(\gamma_3))$, $(\gamma_5, \Phi(\gamma_5))$, . . . расположатся на кривой $y = \Phi(x)$.

Рис. 3.1. Примеры функций Ф(х)

Следовательно, «хорошую» последовательность может породить только такая функция $y = \Phi(x)$, график которой весьма плотно заполняет единичный квадрат. Примером такой функции может служить функция $y = Д(gx)$ при очень больших g (рис. 3.1*б*). И действительно, эта функция послужила основой для ряда методов получения псевдослучайных чисел. Конечно, приведенное условие только необходимо, но далеко недостаточно для того, чтобы формула $\gamma_{n+1} = \Phi(\gamma_n)$ порождала «хорошие» псевдослучайные числа.

Другая важная черта алгоритмов вида $\gamma_{n+1} = \Phi(\gamma_n)$ состоит в том, что при реализации их на компьютере они всегда порождают периодические последовательности. В самом деле, так как в коде любого компьютера можно записать лишь конечное число N чисел, заключенных между нулем и единицей, то рано или поздно какое-нибудь значение γ_L совпадает с одним из предыдущих значений γ_l, необязательно с γ_0 в силу неоднозначности функции $\Phi(x)$. Тогда

$$\gamma_{L+i} = \gamma_{l+i} \quad i=1,2, \ldots .$$

Пусть L – наименьшее число, удовлетворяющее этому соотношению при некотором l ($l \le$ L). Множество чисел $\gamma_0, \gamma_1, ..., \gamma_{L-1}$ называется *отрезком апериодичности* последовательности, число L – *длиной отрезка апериодичности,* а $P = L - l$ – *длиной периода*.

В рассматриваемом случае отрезок апериодичности состоит из различных чисел. И обычно для расчета не рекомендуют использовать больше, чем L чисел последовательности. Ясно также, что $L<N$.

Для экспериментального определения значений L и P можно использовать алгоритм, подробно описанный в [2].

Методы получения псевдослучайных чисел

Рассмотрим несколько конкретных методов, в основе которых рекуррентные формулы первого порядка.

Метод середины квадрата

Это первый алгоритм для получения псевдослучайных чисел, и предложен он был Нейманом. В этом методе число γ_n предполагается $2k$-значным:

$$\gamma_n = 0, a_1 a_2 ... a_{2k}.$$

Чтобы получить число γ_{n+1}, надо γ_n возвести в квадрат. При этом количество значащих цифр результата будет равно $4k$. Затем надо отобрать средние $2k$ цифр этого квадрата :

$$\gamma_{n+1} = 0, b_{k+1} b_{k+2} ... b_{3k}.$$

Нетрудно проверить, что этому методу соответствует функция

$$\Phi(x) = Д[10^{-2k}\, Ц(10^{3k}\, x^2)]$$

или, что то же самое,

$$\Phi(x) = 10^{-2k}\, Ц[10^{2k}\, Д(10^{k}\, x^2)].$$

Однако от метода середины квадрата вычислители отказались, так как в последовательностях, построенных таким образом, получается больше, чем нужно малых чисел. Интересно, что при некоторых γ_0 наблюдается вырождение последовательности, т. е. $\gamma_n \equiv 0$ при $n \ge n_0$.

Метод вычетов

... или метод сравнения (congruential method). Этот метод, предложенный Д. Лемером, получил наибольшее распространение. В этом методе $\Phi(x) = Д(gx)$, т. е.

$$\gamma_{n+1} = Д(g\gamma_n),$$

где g – большое целое число.

Можно доказать, что если задать γ_0 в форме несократимой дроби $\gamma = m_0/M$, где m_0 и M – целые числа, и M взаимно просто с g, то все γ_n будут несократимыми дробями вида $\gamma_n = m_n/M$, где числители m_n определяются формулой:

$$m_{n+1} = gm_n (\text{mod } M).$$

Эта запись означает, что m_{n+1} равно остатку, полученному при делении gm_n на M (или, другими словами, m_{n+1} – это наименьший положительный вычет gm_n по модулю M).

Изучению последовательностей m_n посвящено много работ. Методами теории чисел удалось исследовать длину отрезка апериодичности и оценить некоторые величины, аналогичные коэффициентам корреляции между γ и $\eta = Ф(\gamma)$, между γ и $\eta = Ф(Ф(\gamma))$ и т. п. Вопрос о пригодности таких псевдослучайных чисел в конечном счете решается обычными статистическими тестами, т. е. эмпирически. При некоторых (g, M, m_0) получаются удовлетворительные последовательности, при других – плохие.

На практике обычно используют формулу $m_{n+1} = gm_n (\text{mod } M)$, причем значение M чаще всего выбирают равным максимальному целому числу, которое может быть представлено на моделирующем компьютере. В этом случае операция получения наименьшего положительного вычета gm_n по модулю M выполняется автоматически. Например, на современных тридцатидвухразрядных компьютерах максимальное целое число, для представления которого используется 4 байта, равно 2147483647. Если произведение gm_n превышает это значение, т. е. для двоичного представления gm_n необходимо более 32-х двоичных разрядов, то при записи этого значения в 4 байта памяти сохранятся только 32 младших разряда, т. е. как раз остаток от деления gm_n на M = 2147483647.

Пример датчика псевдослучайных чисел

Для практического применения на современных компьютерах в программах на языке «Си» можно использовать следующую функцию bsv для получения псевдослучайных чисел.

```
   double slch(long *ix)
     { long r;
     r=*ix*65539L;          //или другой вариант r =
*ix * 1220703125;
```

```
    if (r<0) r=r^0x80000000UL;    //или r = r +
2147483647L + 1;
    *ix = r;    return(r*0.4656613e-9);
    }
```

Функция `slch` является аналогом функции *RANDU* из библиотеки процедур, которая использовалась при написании программ на языке FORTRAN в системе *IBM/*360.

В вызывающей эту функцию программе должна быть определена переменная типа long для хранения очередного значения m_n, вычисляемого по методу вычетов. Перед первым обращением к функции `slch` этой переменной должно быть присвоено начальное значение m_0, так называемое, *исходное число датчика*, которое определяет всю последовательность псевдослучайных чисел. Исходное число должно быть любое, но обязательно нечетное и не кратное 5. Однако последовательность псевдослучайных чисел для каждого исходного числа необходимо проверять.

В функцию `slch` передается указатель на m_n. Число m_n умножается на g = 65539 или g = 1220703125 (оба значения дают неплохие последовательности случайных чисел). Результат умножения присваивается переменной *r*. При этом сохраняются только младшие 32 разряда произведения. Поскольку один двоичный разряд определяет знак числа, то в результате такого сохранения может получиться отрицательное число. Если действительно получится отрицательное число, то изменяем этот разряд так, чтобы число стало положительным. Для этого в функции используется оператор `r = r^0x80000000UL`. В результате таких вычислений получается значение *r,* равное m_{n+1}, которое запоминается и в дальнейшем используется при следующем обращении к датчику. Датчик возвращает значение m_{n+1}, деленное на 2147483647 (или умноженное на $0.4656613*10^{-9} \approx 1/2147483647$), т. е. значение из интервала (0,1].

Проверка случайных чисел

Как уже неоднократно упоминалось, случайные числа, полученные любым из трех рассмотренных методов, необходимо проверить. Существует достаточно много различных критериев и методов такой проверки, но их рассмотрение выходит за рамки данного учебного пособия. Это задачи математической статистики. Более того, о достаточности тех или иных критериев можно говорить, только

ограничив класс задач, которые предполагается решать с помощью проверяемых случайных чисел. Здесь рассмотрим лишь самый простой метод, который имеет смысл применять всегда на начальном этапе поверки.

Разобьем отрезок [0,1] на n равных интервалов. Возьмем достаточно большое число N случайных чисел $\gamma_0, \gamma_1, ..., \gamma_N$ и подсчитаем количество ν_i попаданий случайных чисел в каждый из n интервалов. Если все значения ν_i приблизительно равны между собой, то можно считать, что случайные числа равномерно распределены. Далее для более детальной проверки можно применить и другие методы проверки, например, те что приведены в [2]. Если значения ν_i сильно отличаются друг от друга и с увеличением N разница не уменьшается, то числа $\gamma_0, \gamma_1, ..., \gamma_N$ не являются случайными числами и нет смысла проверять их другими более сложными и точными методами.

Преобразования случайных величин

Как уже отмечалось, имитация различных случайных событий, величин, процессов, функций любой сложности сводится к функциональному преобразованию базовых случайных величин. Далее в качестве базовых будем использовать случайные числа $\gamma_0, \gamma_1, \ldots, \gamma_n \ldots$ Рассматривая примеры, будем считать, что мы умеем получать базовые значения, т. е. есть некоторый датчик псевдослучайных чисел, который реализован на языке “Си” в виде отдельной функции с описанием

```
double bsv(void);
```

При каждом обращении к этой функции она возвращает очередное случайное число.

Имитация случайных событий

Пусть необходимо имитировать некоторое событие A, которое наступает с заданной вероятностью p. Для имитации такого события достаточно одного случайного числа γ. Если $\gamma < p$, то можно считать, что событие A наступило, в противном случае, наступило противоположенное событие.

Например, при моделировании некоторые действия должны выполняться при наступлении события A и совершенно другие, если это событие не наступает. Фрагмент программы, моделирующий эту ситуацию, может быть таким:

```
   float p=6.54;            //вероятность наступления
события

   . . .

   if (bsv()<p) { . . . } //действия при наступлении
события A

   else         { . . . } //действия, если событие
не наступает
```

Если необходимо имитировать два или больше событий и заданны вероятностями их наступления, то можно повторять этот прием соответствующее число раз. Лучше это делать с помощью методов имитации дискретных случайных величин.

Имитация дискретных случайных величин

Рассмотрим дискретную случайную величину ξ с распределением

$$\begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix},$$

где p_i – вероятность того, что случайная величина ξ примет значение a_i.

Для имитации ξ разделим интервал (0,1] на интервалы Δ_i длины p_i, i=1,2,...,n:

$$\Delta_1 = (0, p_1]; \quad \Delta_2 = (p_1, p_1+p_2]; \ \ldots \ \Delta_n = (p_1+p_2+\ldots+p_{n-1}, 1].$$

Возьмем одно случайное число γ и проверим, какому из промежутков оно будет принадлежать. Если γ будет принадлежать Δ_k, то следует считать, что случайная величина принимает значение a_k.

Для практической реализации этого метода имитации величины ξ удобно в память хранить (кроме возможных значений случайной величины) значения $y_1 = p_1$, $y_2 = p_1+p_2$, $y_3 = p_1+p_2+p_3$, ..., $y_{n-1} = p_1+\ldots+p_{n-1}$. В этом случае для имитации дискретной случайной величины достаточно γ сравнивать последовательно с y_1, y_2, … y_{n-1}, пока оно не станет меньше очередного из них.

Например, функция, имитирующая дискретную случайную величину, может быть такой:

```
   float dsv (void)  {     //имитация дискретной
случайной величины

     float a[]={5.23, 3,   8,   6,    2.1};//
возможные значения
```

```
    float y[]={0.1,  0.4, 0.7, 0.95, 1  };     //
суммы вероятностей
    double x;
    int i;
    x=bsv( );
    for (i=0; x>y[i]; i++);
    return(a[i]);
    }                              //конец функции dsv
```

Так как порядок значений $a_1, a_2, \ldots a_n$ случайной величины ξ произволен, то выгодно располагать их в порядке убывания вероятностей. При этом среднее количество повторений цикла при получении одного значения случайной величины, очевидно, будет наименьшим.

Имитация дискретной случайной величины значительно упрощается в случае, когда ее значения равновероятны, т. е. $p_1 = p_2 = \ldots = p_n = 1/n$. Действительно, при этом нет необходимости последовательно проверять, какому из интервалов Δ_i принадлежит случайное число γ. Интервал Δ_i есть $((i-1)/n, i/n]$, $i=1,2,\ldots,n$, и принадлежность случайного числа γ этому интервалу равносильно условию $i - 1 \le \gamma n \le i$. Отсюда $i = Ц[n\gamma] + 1$, где $Ц[n\gamma]$ обозначает целую часть числа. Следовательно, для имитации значения дискретной случайной величины с равновероятными значениями достаточно взять одно случайное число и по формуле $i = [n\gamma] + 1$ вычислить номер значения случайной величины.

Имитацию случайных событий можно легко свести к имитации соответствующих дискретных случайных величин. Действительно, если известно, что случайное событие A имеет вероятность p, то имитация этого события по рассмотренному ранее алгоритму есть почти то же самое, что имитация дискретной случайной величины, которая принимает значение 1 с вероятностью p и значение 0 с вероятностью $1 - p$. Такая случайная величина называется индикатором события A.

Если необходимо имитировать наступление одного из попарно несовместимых событий $A_1, A_2, \ldots, A_n$, имеющих вероятности $p_1, p_2, \ldots, p_n$, то можно ввести случайную величину, которая с этими вероятностями принимает значения номеров соответствующих событий.

При имитации совместимых событий можно поступать двояко. Если, например, надо имитировать два совместимых события A и B, вероятности наступления которых p_A и p_B заданы, то можно взять два базовых значения и рассмотренным ранее методом по одному из них

имитировать событие A, а по другому – B. А можно поступать иначе. Взять четыре попарно несовместимых события:

$$A_1 = AB, \quad A_2 = A\overline{B}, \quad A_3 = \overline{A}B, \quad A_4 = \overline{A}\,\overline{B},$$

(черта над символом события означает противоположенное событие), вычислить их вероятности:

$$p_1 = p_A p_B, \quad p_2 = p_A(1-p_B), \quad p_3 = (1-p_A)p_B, \quad p_4 = (1-p_A)(1-p_B),$$

с помощью одного базового значения имитируем соответствующую этим четырем событиям дискретную случайную величину и по ее значению решить, какие события наступили.

Совершенно аналогично можно имитировать наступление двух зависимых совместимых событий A и B, для которых заданы вероятности p_A, p_B и p_{AB}. При этом надо рассматривать такую же полную группу попарно несовместимых событий, что и в предыдущем случае, только вероятности этих событий будут другие:

$$p_1 = p_{AB}, \quad p_2 = p_A - p_{AB}, \quad p_3 = p_B - p_{AB}, \quad p_4 = 1 - p_A - p_B + p_{AB}.$$

Имитация непрерывных случайных величин

Пусть случайная величина ξ определена на некотором интервале $[a,b]$ и имеет на этом интервале плотность $p(x) > 0$. Обозначим через $F(x)$ функцию распределения ξ, которая при $a \le x < b$ равна:

$$F(x) = \int_a^x p(u)du$$

(случай бесконечных значений a и b не исключается).

Метод обратных функций

Этот метод имитации основан на следующем утверждении. Если γ – случайная величина с равномерным распределением на (0,1), то случайная величина ξ, удовлетворяющая уравнению:

$$F(\xi) = \gamma,$$

имеет плотность распределения $p(x)$.

Действительно, т.к. $F(x)$ возрастает на $[a,b]$ от 0 до 1, то приведенное выше уравнение имеет единственный корень при каждом γ. При этом равны вероятности;

$$\mathrm{P}\{x < \xi < x + \mathrm{d}x\} = \mathrm{P}\{F(x) < \gamma < F(x + \mathrm{d}x)\}.$$

Так как случайная величина γ равномерно распределена на (0,1), то

$$P\{x < \xi < x + dx\} = F(x + dx) - F(x) = p(x)dx,$$

что и требовалось доказать.

В тех случаях, когда рассматриваемое уравнение аналитически разрешимо относительно ξ, получается явная формула для вычисления этой случайной величины:

$$\xi = F^{-1}(\gamma),$$

где $F^{-1}(\gamma)$ – обратная функция по отношению к $F(x)$.

Метод обратных функций позволяет записать формулы для имитации любой случайной величины, но нередко этот метод приводит к сложным или просто неудобным алгоритмам. Например, чтобы вычислить значение гауссовской случайной величины с параметрами (0,1), приходится решать уравнение:

$$\int_{-\infty}^{\xi} e^{-\frac{t^2}{2}} dt = \sqrt{2\pi}\gamma .$$

Это не простая задача. В таких случаях уравнение $F(\xi) = \gamma$ решают численно, либо составляют таблицу $F^{-1}(\gamma)$ и с помощью интерполяции по ней находят значения ξ. Чаще всего прибегают к помощи других методов имитации, связанных с другими преобразованиями γ.

Метод исключения

Метод исключения (метод отбора, метод режекции) [3] предусматривает следующую процедуру получения реализации x случайной величины ξ, с плотностью вероятностей $p(x) = g(x)$.

1) Выбирается некоторая можарирующая кривая $g_1(x) \geq g(x)$ (рис. 3.2).

Рис. 3.2. Метод исключения

2) Имитируется реализация $[x, y]$ случайного вектора $[\xi', \eta']$, равномерно распределенного в любой области G_1, ограниченной осью x и мажорирующей кривой $g_1(x)$.

3) Если $y < g(x)$, то x есть искомая реализация ξ. В противном случае реализация $||x, y||$ отбрасывается, и этапы (б), (в) повторяются.

Эффективность метода исключения характеризуется коэффициентом использования, т. е. отношением среднего числа

полученных реализаций величины ξ к числу «затраченных» реализаций [pξ',η'] или отношением площади области G к площади области G_1.

Частным случаем метода исключений является метод Неймана, когда случайная величина ξ определена на отрезке (a, b) и имеет ограниченную плотность распределения $p(x) \le c$. Тогда в качестве области G_1 можно взять прямоугольник $\Pi = \{a < y_1 < b,\ 0 < y_2 < c\}$, (рис. 3.3). Для получения реализации [x, y] случайного вектора [ξ', η'], равномерно распределенного в прямоугольнике Π, получаем два случайных числа γ_1, γ_2, и вычисляем координаты x и y по формулам:

$$x = a + \gamma_1(b - a), \qquad y = c\gamma_2.$$

Если $y < p(x)$, то в качестве искомого значения ξ берем x. Иначе получаем два новых случайных числа и повторяем процедуру.

Рис. 3.3. Метод Неймана

Для обоснования данного алгоритма вычислим условную вероятность:

$$P\{\xi < z\} = P\{x < z \,/\, y < p(x)\} = \frac{P\{x < z, y < p(x)\}}{P\{y < p(x)\}}.$$

Знаменатель последнего выражения есть вероятность того, что точка с координатами [x, y] попадает под кривую $p(x)$. Так как плотность распределения случайной точки [x, y] постоянна и равна 1/[c ($b - a$)], то

$$P\{y < p(x)\} = \int_a^b dq \int_0^{p(q)} \frac{dw}{c(b-a)} = \frac{1}{c(b-a)} \int_a^b p(q)dq = \frac{1}{c(b-a)}.$$

Если учесть, что

$$P\{x < z,\ y < p(x)\} = \int_a^z dq \int_0^{p(q)} \frac{dw}{c(b-a)} = \frac{1}{c(b-a)} \int_a^z p(x)dx.$$

Таким образом получаем

$$P\{\xi < z\} = \int_a^z p(x)dx,$$

что и требовалось доказать.

Эффективность метода Неймана, очевидно, будет равна вероятности попадания точки с координатами $[x, y]$ под кривую $p(x)$, а эта вероятность равна отношению площади под кривой к площади прямоугольника $\Pi = \{a < y_1 < b,\ 0 < y_2 < c\}$. Следовательно, эффективность метода будет наибольшей, если выбрать наименьшее возможное c. Метод Неймана не эффективен, когда $p(x)$ имеет явно выраженный максимум. В этом случае необходимо выбирать другую мажорирующую функцию, например, ступенчатую.

Метод суперпозиции

Пусть функция распределения $F(x)$ интересующей нас случайной величины ξ представима в виде суммы функций распределения $F_k(x)$:

$$F(x) = \sum_{k=1}^{m} c_k F_k(x),$$

в которой коэффициенты c_k все положительны и в сумме равны 1. В этом случае можно ввести дискретную случайную величину η с распределением:

$$\begin{pmatrix} 1 & 2 & \cdots & m \\ c_1 & c_2 & \cdots & c_m \end{pmatrix},$$

так что $P\{\eta = k\} = c_k$.

Для имитации случайной величины ξ методом суперпозиции возьмем два случайных числа γ_1, γ_2. С помощью γ_1 имитируем случайную величину η. Пусть в результате имитации получим $\eta = k$. Затем из уравнения $F_k(\xi) = \gamma_2$ определим искомое значение ξ. Действительно, по теореме о полной вероятности

$$P\{\xi < x\} = \sum_{k=1}^{m} P\{\xi < x / \eta = k\} P\{\eta = k\} = \sum_{k=1}^{m} F_k(x) c_k = F(x).$$

Возможность обобщения метода суперпозиции на случай бесконечного числа слагаемых в представлении функции распределения в виде сумме и на многомерные распределения очевидна.

Функции распределения, которые можно представить в виде суммы функций распределения, встречаются тогда, когда мы имеем дело со смесью случайных величин. Если некоторую функцию распределения можно представить в такой форме, то этим можно воспользоваться для упрощения процедуры имитации.

Например, случайная величина ξ определена на интервале [0,1] и имеет функцию распределения:

$$F(x) = \sum_{k=1}^{\infty} c_k x^k, \quad c_k > 0, \quad k = \overline{1,\infty}.$$

Мы можем считать, что $F_k(x) = x^k$ при $0 < x < 1$. Воспользуемся методами суперпозиции и обратных функций. Возьмем два случайных числа γ_1 и γ_2. Если

$$\sum_{j=1}^{k-1} c_j \le \gamma_1 < \sum_{j=1}^{k} c_j \quad , \quad \text{то} \quad \xi = (\gamma_2)^{1/k}.$$

Другой пример. Случайная величина ξ определена в интервале [0,2] с плотностью $p(x)$ и, соответственно, функцией распределения $F(x)$:

$$p(x) = \frac{5}{12}[1 + (x-1)^4]; \qquad F(x) = \frac{5}{12}[x + \frac{1}{5} + \frac{1}{5}(x-1)^5].$$

Если для имитации ξ использовать метод обратных функций, то надо решать уравнение пятой степени $(\xi - 1)^5 + 5\xi = 12\gamma - 1$. Но $p(x)$ можно представить в виде суперпозиции плотностей $p_1(x) = ½$ и $p_2(x) = 5(x - 1)^4/2$:

$$p(x) = \frac{5}{6} p_1(x) + \frac{1}{6} p_2(x),$$

по методу суперпозиции получаем явный алгоритм

$$\xi = \begin{cases} 2\gamma_2 & \text{åñëè } \gamma_1 < 5/6, \\ 1 + \sqrt[5]{2\gamma_2 - 1} & \text{åñëè } \gamma_1 \ge 5/6. \end{cases}$$

Имитация величин с часто встречающимися распределениями

Рассмотрим некоторые из алгоритмов имитации случайных величин с известными распределениями, которые приведены в книге [3]. В этой же книге достаточно обширная библиография по алгоритмам и методам моделирования случайных величин с различными распределениями.

Нормальное распределение

Нормально распределенная величина ξ' с математическим ожиданием α и дисперсией σ^2 определяется плотностью вероятностей:

$$p_{\xi'}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\alpha)^2}{2\sigma^2}}, \quad -\infty < x < \infty$$

и имеет функцию распределения

$$F(\xi' \le x) = \Phi\left(\frac{x-\alpha}{\sigma}\right), \quad \text{где} \quad \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{x^2}{2}} dx.$$

Эта величина может быть получена в виде

$$\xi' = \alpha + \sigma\xi,$$

где ξ нормально распределенная величина с нулевым математическим ожиданием и единичной дисперсией, т. е. с параметрами $\alpha = 0$, $\sigma = 1$. Рассмотрим несколько способов имитации ξ.

Первый способ использует метод обратных функций

$$\xi = \begin{cases} \tilde{\Phi}^{-1}(\gamma), & 0{,}5 < \gamma < 1, \\ -\tilde{\Phi}^{-1}(1-\gamma), & 0 < \gamma \le 0{,}5, \end{cases}$$

где функция $\tilde{\Phi}^{-1}(x)$, аппроксимирующую обратную к функции распределения $\Phi(x)$, может быть задана в виде

$$\tilde{\Phi}^{-1}(\gamma) = \frac{2{,}30753 + 0{,}27061\psi}{1 + 0{,}99229\psi + 0{,}04481\psi^2}$$

(с ошибкой $3 \cdot 10^{-3}$) или

$$\tilde{\Phi}^{-1}(\gamma) = \psi - \frac{2{,}515517 + 0{,}802853\psi + 0{,}010328\psi^2}{1 + 1{,}432788\psi + 0{,}189269\psi^2 + 0{,}001308\psi^3}$$

(с ошибкой $4{,}5 \cdot 10^{-4}$).

В этих формулах $\psi = \sqrt{-2\ln\gamma}, \quad 0{,}5 < \gamma < 1$.

Второй способ использует преобразование пары независимых случайных чисел γ_1 и γ_2 в пару независимых нормальных величин:

$$\xi_1 = \sqrt{-2\ln\gamma_1}\cos 2\pi\gamma_2,$$
$$\xi_2 = \sqrt{-2\ln\gamma_1}\sin 2\pi\gamma_2.$$

Этот способ чувствителен к корреляции чисел γ_1 и γ_2, поэтому имеет смысл использовать различные датчики для получения γ_1 и γ_2. Для вычисления логарифма и тригонометрических функций целесообразно использовать простые и быстрые алгоритмы, а не стандартные функции, которые дают высокую точность, излишнюю при имитации случайных величин.

Третий способ использует свойство сходимость сумм независимых величин к нормальному распределению. Формула

$$\xi = \sqrt{\frac{12}{n}}\left[\sum_{i=1}^{n}\gamma_i - \frac{n}{2}\right]$$

имеет при $n = 12$ погрешности, не превышающие $9\cdot10^{-3}$ для $|\xi| < 2$ и $9\cdot10^{-1}$ для $2 < |\xi| < 3$. Для многих приложений оказывается достаточным брать $n = 6$.

Экспоненциальное распределение

Экспоненциально распределенная случайная величина ε с параметром λ определяется плотностью распределения

$$p(x) = \lambda e^{-\lambda x}, \quad x \geq 0$$

и функцией распределения

$$F(x) = 1 - e^{-\lambda x}.$$

Эту случайную величину удобно имитировать методом обратных функций, поскольку уравнение $F(\xi) = \gamma$ имеет вид $1 - e^{-\lambda x} = \gamma$. Отсюда легко находится в явном виде выражение для ξ:

$$\xi = -\frac{1}{\lambda}\ln(1-\gamma).$$

Для имитации значения экспоненциально распределенной случайной величины ξ необходимо взять одно случайное число γ, и вычислить значение случайной величины ξ по формуле

$$\xi = -\frac{1}{a}\ln\gamma.$$

При этом учтено, что случайная величина $1-\gamma$ имеет такое же распределение, как и γ.

Второй способ основан на методе исключения. Последовательно имитируется 1-я серия случайных чисел $\gamma_i^{(1)}$, $i = 1,2,\ldots$. Имитация продолжается до тех пор, пока $\gamma_1^{(1)} > \gamma_2^{(1)} > \ldots > \gamma_n^{(1)}$, но $\gamma_{n+1}^{(1)} \geq \gamma_n^{(1)}$. Если n – нечетно, то искомая реализация экспоненциально распределенной случайной величины ε равна $\gamma_1^{(1)}$. Если n четно, то имитируется новая, не зависимая от первой, серия случайных чисел $\gamma_1^{(2)} > \gamma_2^{(2)} > \ldots > \gamma_n^{(2)}$. Так продолжается до тех пор, пока j-я серия $\gamma_i^{(j)}$, $i = 1,2,\ldots$ не закончится на числе $\gamma_n^{(j)}$ с нечетным номером n. При этом

$$\varepsilon = j - 1 + \gamma_1^{(j)}.$$

В среднем для имитации одной реализации ε затрачивается примерно 6 случайных чисел, что в ряде случаев выполняется быстрее, чем при использовании метода обратных функций с вычислением логарифма по стандартной программе.

Третий способ использует метод композиции. Имитируются две независимые дискретные случайные величины τ и θ с распределениями:

$$P\{\tau = m\} = (e-1)e^{-m}, \qquad m = 1, 2, \ldots,$$

$$P\{\theta = n\} = \frac{1}{(e-1)n!}, \qquad n = 1, 2, \ldots,$$

при этом

$$\varepsilon = \tau - \max(\gamma_1, \gamma_2, \ldots, \gamma_\theta).$$

Несмотря на кажущуюся сложность, способ достаточно быстрый и требует для одной реализации ε в среднем 3 или 4 случайных числа.

Распределение Релея

Плотность распределения Релея имеет вид

$$p(x) = \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}, \quad x \geq 0.$$

Значение случайной величины ν с этим распределением можно получить по одной из следующих формул:

$$\nu = \sigma\sqrt{-2\ln\gamma} = \sigma\sqrt{\xi_1^2 + \xi_2^2} = \sigma\sqrt{2\varepsilon},$$

где γ – случайное число, ξ_1 и ξ_2 – независимые значения нормальной случайной величины с нулевым математическим ожиданием и единичной дисперсией, ε – значение экспоненциальной случайной величины с параметром распределения $\lambda = 1$.

Гамма-распределение

Это распределение используется при имитации случайных потоков Эрланга, которые в теории массового обслуживания часто выбираются в качестве моделей входных потоков заявок. Плотность гамма-распределения записывается в виде

$$p(x) = \frac{1}{\Gamma(n)} \lambda^n x^{n-1} e^{-\lambda x}, \quad x \geq 0,$$

где $\Gamma(n)$ – гамма-функция, n – целое положительное число, λ – параметр распределения.

Формула для вычисления значения случайной величины ν с гамма-распределением имеет вид

$$\nu = \frac{1}{\lambda} \sum_{i=1}^{n} \varepsilon_i,$$

где ε_i – независимые значения экспоненциальной случайной величины с параметром распределения $\lambda = 1$.

Имитация величин со ступенчатыми плотностями

Ступенчатая плотность вероятностей является удобной и простой аппроксимацией для плотностей сложной формы. Кроме того, ступенчатая плотность используется для представления в модели входных данных, заданных в виде экспериментально снятых гистограмм.

Итак, пусть плотность распределения случайной величины ξ задана на интервале $[a, b]$ в виде ступенчатой функции $p(x)$ (рис. 3.4*а*). Высота i-й ступеньки равна c_i, ширина $x_i - x_{i-1}$, $i = 1, 2, \ldots, n$; $x_0 = a$; $x_n = b$. Вероятность p_i того, что ξ примет значение x из промежутка (x_{i-1}, x_i) равна

$$P\{x_{i-1} < x < x_i\} = p_i = c_i (x_i - x_{i-1}), \qquad i = 1, 2, \ldots, n.$$

Функция распределения случайной величины ξ $F(x) = P\{\xi < x\}$ (рис. 3.4*б*) при $x \in [x_i, x_{i-1}]$ равна

$$y = F(x) = \int_a^x p(u)du = \int_{x_0}^{x_1} c_1 du + \int_{x_1}^{x_2} c_2 du + \cdots + \int_{x_{i-1}}^{x} c_i du =$$

$$= \sum_{k=1}^{i-1} c_k (x_k - x_{k-1}) + c_i (x - x_{i-1}) = d_{i-1} + c_i (x - x_{i-1}),$$

где $d_i = \sum_{k=1}^{i} c_k (x_k - x_{k-1})$.

При этом y принимает значения из интервала $[d_{i-1}, d_i]$. Следовательно, обратная функция $x = F^{-1}(y)$ на интервале $[d_{i-1}, d_i]$ равна

$$x = F^{-1}(y) = \frac{y - d_{i-1}}{c_i} + x_{i-1} \quad \text{или} \quad x = \frac{x_i - x_{i-1}}{p_i}(y - d_{i-1}) + x_{i-1}$$

и принимает значения из интервала $[x_i, x_{i-1}]$.

Рис. 3.4. Имитация величин со ступенчатыми плотностями

Таким образом, для того, чтобы имитировать случайную величину ξ, берем одно случайное число γ. Определяем, в какой промежуток $[d_i, d_{i-1}]$ это число попадает, т. е. определяем номер промежутка i. Для этого используем процедуру имитации дискретной случайной величины, принимающей значение i с вероятностью p_i. Затем по найденному i вычисляем искомое значение случайной величины ξ по формуле

$$\xi = \frac{x_i - x_{i-1}}{p_i}(\gamma - d_{i-1}) + x_{i-1}.$$

Глава 4

Примеры моделирования

Рассмотрим несколько примеров описания систем массового обслуживания и создания главных функций имитационной модели для программ моделирования этих систем на языке «Си». Будем использовать только простейшие средства языка. Для упрощения текстов функций будем предполагать, что для всех параметров СМО, переменных модели, векторов состояний и событий в программах определены соответствующие глобальные константы, переменные и массивы. В частности константу *BigC* со значением 10^{300} определим директивой

```
#define BigC 1E+300
```

Во всех программах будем использовать следующие переменные.

```
double t;       //модельное время
double tnew;        //момент наступления следующего события
int ms, msnew;      //индексы текущего и следующего событий
```

Вектор событий во всех программах будем определять как массив

```
double tau[K];      //вектор событий
```

Константу *K* со значением количества потоков основных событий также будем использовать во всех программах и задавать директивой #define

Относительно моделирования длительностей локальных процессов будем предполагать, что есть некие функции, при обращении к которым они возвращают длительности очередных локальных процессов.

Будем также считать, что для моделирования случайных событий и величин есть функция

```
double bsv(void);       //датчик случайных чисел,
```

При каждом обращении к ней она возвращает новое значение случайной величины, равномерно распределенной на интервале [0,1].

Будем предполагать, что функции «Ввод параметров», «Планирование эксперимента» и «Программа эксперимента», которые представлены на блок-схеме главной функции программы моделирования на рис. 2.8, определены и описаны следующим образом:

```
void param(void);
void planexp(void);
int exper(void);
```

Тексты этих функций приводить не будем, поскольку написание их не представляет труда даже для неопытных программистов.

С учетом сделанных предположений главная функция для всех программ моделирования может быть записана в виде:

```
void main(void)
  { int i;     //рабочая переменная
  param();     //ввод параметров
  planexp();       //планирование эксперимента
C:                 //начало цикла моделирования
  {
      //Блок «Модель» – обращение к функциям
обработки событий
  }
      //момент наступления и индекс следующего
событияи
  tnew = BigC;
  for (i=0;i<K;i++)
    if (tau[i]<tnew) {tnew=tau[i]; msnew=i;}
  i=exper();       //обращение к функции
Обработка результатов
      //проверяем условие окончания эксперимента
  if (i>0) {       //если эксперимент необходимо
продолжить, то
    t=tnew; ms=msnew;//времени и индексу события
новые значения и
    goto C;}       //к началу цикла моделирования
  }   //конец программы
```

Блок «Модель» должен содержать операторы обращения к функциям обработки событий в зависимости от индекса наступившего события, т. е. от значения переменной `ms`.

В качестве объектов моделирования выберем хорошо известные всем реальные системы.

Минимаркет

Рассмотрим магазин по продаже штучных товаров с небольшим торговым залом и одним продавцом. Покупатели в таком магазине сначала рассматривают товар на витринах и, если выбирают что-либо, по очереди подходят к продавцу, рассчитываются за покупку и уходят. У магазина есть один или несколько поставщиков товаров. Есть график, по которому продавец по телефону сообщает поставщику, какие товары и в каком количестве надо доставить в магазин. Через определенное этим графиком время товар доставляется, продавец прерывает обслуживание покупателей и принимает товар. После этого работа магазина продолжается в обычном режиме.

Создадим несколько простых систем массового обслуживания этого магазина. Начнем с наипростейшей первой версии.

СМО «Минимаркет». Версия 1

Будем считать заявками только покупателей магазина. Никаких различий между покупателями учитывать не будем, т. е. все заявки будем считать однотипными и без параметров.

Пусть заявки поступают из внешних, независимых источников по одной единым потоком и задан единый закон $F_{вх}$, определяющий длительности интервалов времени между моментами поступления любых очередных заявок.

Будем считать, что СМО состоит из одного обслуживающего прибора, который будет моделировать работу продавца. В каждый момент времени прибор может обслуживать только одну заявку. Длительность каждого обслуживания определяется некоторым законом, который обозначим $F_{обс}$. Схема СМО изображена на рис. 4.1. На схеме прямоугольник изображает прибор, стрелки указывают пути продвижения заявок, точки – очередь заявок.

Рис. 4.1. СМО «Минимаркет-1»

Дисциплину обслуживания выберем следующую. Каждая поступающая в СМО заявка принимается на обслуживание в прибор, если он в этот момент свободен, т. е. не занят обслуживанием другой заявки. Если прибор занят, поступившая заявка становится в очередь вслед за ранее поступившими заявками. Будем предполагать, что, если в момент поступления заявки в очереди уже есть N заявок, то поступившая заявка не принимается в СМО. В этом случае говорят, что заявка «теряется», а СМО работает с отказами в обслуживании. После обслуживания в приборе заявка покидает СМО, а прибор мгновенно берет на обслуживание следующую заявку из очереди. Если заявок в очереди в этот момент нет, прибор остается свободным, т. е. переходит в состояние ожидания поступления заявки. Для определенности будем считать, что заявка, которая обслуживается прибором, продолжает находиться первой в очереди до окончания обслуживания.

Описание программы моделирования

Параметры

У СМО предусмотрено ограничение на количество одновременно находящихся в системе заявок, которые моделируют покупателей. Количество не должно превышать некоторого значения N. Для определенности зададим эту константу директивой

```
#define N 100      //максимальное количество заявок в СМО
```

Функции моделирования локальных процессов

Будем предполагать, что для моделирования длительностей локальных процессов, которые определяются законами $F_{вх}$ и $F_{обс}$, есть соответствующие функции

```
double fvx(void);  //длительность ожидания заявки
double fobs(void); //длительность обслуживания заявки
```

Вектор состояния

Для описания состояния системы в любой момент времени достаточно одного значения – количество заявок в системе. Это значение определяет не только общее количество заявок в системе, но и состояния очереди и прибора. Если значение равно нулю, т. е. в системе нет заявок, то прибор свободен. Если значение больше нуля, то все

заявки в очереди и первая из них обслуживается в приборе. Опишем в программе переменную оператором

```
int k0;          //количества заявок в системе
```

Вектор событий

Совершенно естественно, что входной поток заявок представляет собой одну последовательность однотипных локальных процессов ожидания заявок и, соответственно, один поток основных событий. Процессы обслуживаний в единственном приборе образуют другую последовательность также однотипных локальных процессов и поток основных событий. Соответственно, константе *K* в программе должно бы присвоено значение 2.

```
#define K 2          //количество потоков основных событий
double tau[K];       //вектор событий
```

Пусть первая компонента вектора `tau[0]` определяет момент поступления очередной заявки, вторая `tau[1]` – момент окончания обслуживания в приборе.

Блок «Модель»

В СМО определены два потока основных событий двух типов, поэтому достаточно двух функций обработки этих событий:

```
void sob0(void);    //поступление заявки
void sob1(void);    //окончание обслуживания в приборе
```

Блок «Модель» также достаточно прост для этой версии СМО.

```
if (ms==0)    sob0();
else          sob1();
```

Функция sob0

Эта функция формирует состояние системы и корректирует вектор событий при каждом поступлении заявки в систему, т. е. при наступлении основного события в потоке 0. Независимо от того, в каком состоянии находится система, это событие является началом нового локального процесса ожидания следующей заявки. Следовательно, первое, что необходимо выполнить в функции `sob0`, так это вычислить момент прихода следующей заявки, т. е. вычислить новое значение `tau[0]` оператором

```
tau[0]=t+fvx();
```

Дальнейшие действия в sob0 зависят от текущего состояния системы. Если в системе уже есть *N* заявок, т. е. значение k0 равно *N*, то функция должна завершить работу, поскольку наступившее основное событие (приход заявки) не влечет за собой в этом случае никаких сопутствующих событий. Заявка в систему не принимается, состояние системы не изменяется, момент окончания локального процесса обслуживания в приборе корректировать не надо. Если же текущее значение k0 меньше *N*, то заявка принимается в систему и становится в очередь к прибору, т. е. наступает сопутствующее событие «прием заявки». Выполнить в sob0 соответствующие этому событию изменения состояния системы можно оператором

```
k0++;
```

Событие «прием заявки» не влечет за собой других событий, если в системе уже есть по крайней мере одна заявка. Если же заявок нет, то принятая заявка становится первой и единственной в очереди и, в соответствии с дисциплиной обслуживания, прибор берет ее на обслуживание. Наступает сопутствующее событие «начало обслуживания». Это событие не изменяет состояние системы, но является началом локального процесса обслуживания. Следовательно, в функции sob0 необходимо вычислить новое значение tau[1], т. е. смоделировать момент окончания этого процесса оператором

```
tau[1]=t+fobs();
```

Таким образом, полностью функцию sob0 можно записать следующим образом:

```
   void sob0(void) {       //поступила заявка на вход системы
     tau[0]=t+fvx();       //момент прихода следующей заявки
     if (k0<N) {           //если условие приема выполняется
       k0++;             //принимаем заявку и ставим в очередь
       if (k0==1)          //если заявок одна, то моделируем...
```

```
        tau[1]=t+fobs();}// момент окончания ее
обслуживания
    }                   //конец sob0
```

Функция sob1

Эта функция выполняется при наступлении основного события в потоке 1, т. е. при окончании обслуживания заявки в приборе. Безусловно, это приводит к уменьшению заявок в системе. Следовательно, функция должна начинаться с оператора, который уменьшает значение `k0` на единицу. Безусловно, также и то, что необходимо вычислить новое значение `tau[1]`. Но каково оно будет, зависит от состояния, в которое перейдет система, т. е. от нового значения `k0`. Если оно больше нуля, то в системе остались еще заявки и по дисциплине обслуживания прибор должен взять первоочередную заявку на обслуживание. В этом случае момент окончания обслуживания должен вычисляться оператором

`tau[1]=t+fobs();.`

Если же в системе не осталось заявок, то новый локальный процесс обслуживания не начнется и в этом случае `tau[1]` надо присвоить значение `BigC`.

Кроме перечисленных, других событий в системе дисциплиной обслуживания не предусмотрено, следовательно, функцию `sob1` можно окончательно записать в виде:

```
   void sob1(void) {       //закончилось обслуживание
в приборе
     k0--;                 //обслуженная заявка ушла
из системы
     if (k0>0)             //если в системе остались
заявки, то
       tau[1]=t+fobs();        //начинаем новое
обслуживание
     else                  //если заявок в системе не
осталось
       tau[1]=BigC;        //переводим прибор в сост.
ожидания
     }                   //конец sob1
```

На этой модели можно исследовать пропускную способность магазина, изменения длины очереди, время занятости продавца, оценить среднее время пребывания покупателя в магазине и т. д. Но результаты все эти исследования могут оказаться недостаточно точными, поскольку модель не учитывает, например, покупателей, которые уходят из магазина без покупок. Попытаемся учесть это и создадим более сложную, более совершенную модель.

СМО «Минимаркет». Версия 2

Будем считать, что в эту СМО, как и в предыдущей версии, поступает поток однотипных заявок-покупателей, определяемый законом $F_{вх}$.

Определим два узла обслуживания. Первый узел будет состоять из N совершенно одинаковых обслуживающих приборов, которые будут моделировать места, занимаемые покупателями в торговом зале магазина. Длительность обслуживания любой заявки в любом из этих приборов есть случайная величина с заданной функцией распределения $F_{выб}$. Второй узел будет состоять из одного прибора и очереди заявок перед ним. Как и в первой версии, этот прибор будет моделировать работу продавца и очередь к нему. Случайная длительность обслуживания заявок в этом приборе пусть определяется некоторой заданной функцией распределения $F_{обс}$. Схема СМО второй версии изображена на рисунке 4.2.

Рис. 4.2. СМО «Минимаркет-2»

Определим следующую дисциплину обслуживания. Каждая поступающая заявка сразу начинает обслуживаться в первом из свободных приборов первого узла обслуживания. Если все приборы первого узла заняты, заявка теряется. После обслуживания в приборе первого узла заявка с вероятностью p покидает СМО и с вероятностью $1–p$ поступает на обслуживание в прибор второго узла. Если заявка покидает СМО, то обслуживший её прибор освобождается для приема очередной поступающей в систему заявки. Если заявка остается в СМО, то обслуживший ее прибор первого узла считается занятым до тех пор, пока заявка не покинет СМО после обслуживания во втором приборе. Таким образом, первый узел обслуживания моделирует поведение покупателей, которые заходят в магазин, если там не слишком много (меньше N) покупателей, рассматривают витрины и

решают, купить что-нибудь или не купить. Если ничего не выбирают, то уходят, освобождая место в магазине. Если решаются на покупку, то остаются в магазине, становятся в очередь к продавцу, обслуживаются, оплачивают покупку и уходят.

Обслуживание заявок во втором узле точно такое же, как в СМО первой версии, т. е. второй узел работает также, только заявки в него поступают не из входного потока, а из первого узла обслуживания.

Описание программы моделирования

Параметры

Дисциплину СМО данной версии определяют два параметра: максимальное количество N заявок, которые одновременно могут находиться в системе, и вероятность p ухода заявки из системы после обслуживания в приборе первого узла. Как и раньше N определим как константу директивой

```
#define N 100       //максимальное количество заявок в СМО
```

Для вероятности p введем переменную

```
float p;        //вероятность ухода заявки после первого обслуживания
```

Значение этой переменной должно быть определено в функции `param`.

Функции моделирования локальных процессов

Будем предполагать, что для моделирования длительностей локальных процессов по законам $F_{вх}$, $F_{выб}$ и $F_{обс}$, определены функции

```
double fvx(void);  //ожидание заявки
double fvyb(void); //обслуживание в первом узле
double fobs(void); //обслуживание во втором узле
```

Вектор состояния

Для описания в программе моделирования состояния системы одной переменной $k0$, как она описана в предыдущей версии

```
int k0;         //количества заявок в системе
```

явно недостаточно. По значению $k0$ нельзя определить, сколько приборов первого узла и какие из них заняты, и сколько заявок стоит в очереди ко второму прибору. Эту переменную можно и даже нужно включить в вектор состояния, но надо ввести дополнительные

переменные. Это прежде всего переменные, значения которых бы определяли состояния приборов первого узла. Опишем их оператором

```
int s[N]; //состояния приборов первого узла
```

Если значение `s[i]` равна 0, то это значит, что прибор с номером i +1 свободен. (Приборы будем здесь и далее нумеровать от 1 до N, а индексы элементов массива, как известно, начинаются с нуля.) Значение `s[i]` равное 1 будет означать, что прибор i+1 обслуживает заявку, а значение равное 2 – прибор занят, т. е. заявку прибор обслужил, но она не покинула систему, а находится в очереди к прибору второго узла или уже обслуживается в нем.

Состояние прибора второго узла будем определять, как и в предыдущей версии, количеством заявок в очереди. Для этого введем соответствующую переменную `k2` и опишем ее оператором

```
int k2;          //количество заявок в очереди ко
второму прибору
```

Значение этой переменной можно, конечно, вычислить по значениям других введенных переменных состояния, но она может оказаться весьма полезной при написании программы моделирования.

Вектор событий

В СМО этой версии, как и в первой, входной поток заявок представляет собой одну последовательность однотипных локальных процессов и, соответственно, один поток основных событий.

В первом узле на обслуживание может находиться до N заявок одновременно, поэтому процессы их обслуживания не могут быть включены в одну последовательность локальных процессов, хотя все они однотипные. Процессы обслуживания в одном приборе не пересекаются во времени и образуют последовательность локальных процессов. Следовательно, все однотипные процессы обслуживания в первом узле образуют N последовательностей локальных процессов, N потоков основных событий.

Второй узел состоит из одного прибора. Поэтому все локальные процессы обслуживания в этом приборе образуют одну последовательность, а моменты окончаний обслуживаний – один поток основных событий.

Таким образом, вектор событий для моделирования данной СМО должен содержать $K = 2 + N$ компонент. Константу K определим директивой

```
   #define K 102       //количество потоков основных
событий
   double tau[K];      //вектор событий
```

Назначения компонент выберем следующие: `tau[0]` – момент поступления заявки; `tau[1],...,tau[N]` – окончания обслуживаний в приборах первого узла; `tau[N+1]` – момент окончания обслуживания в приборе второго узла.

Блок «Модель»

Поскольку в СМО возможны основные события только трех типов, то достаточно трех функций их обработки:

```
   void sob0(void);    //поступление заявки
   void sob1(void);    //окончание обслуживания в
первом узле
   void sob2(void);    //окончание обслуживания во
втором узле
```

Блок «Модель» в главной функции программы может быть записан в виде:

```
   if (ms==0)          sob0();
   else if (ms==N+1)   sob2();
        else           sob1();
```

Функция sob0

Эта функция обработки факта появления новой заявки на входе системы, как и одноименная функция моделирования СМО первой версии, должна начинаться с моделирования момента поступления следующей заявки и проверки условия принятия заявки в систему. Если условие выполняется, то необходимо найти свободный прибор первого узла, направить туда заявку и смоделировать момент времени окончания ее обслуживания. Следовательно, функцию можно записать так.

```
   void sob0(void)          //поступила заявка на вход
системы
     {int i;                //рабочая переменная
     tau[0]=t+fvx();        //момент прихода следующей
заявки
```

```
    if (k0<N) {           //если условие приема
выполняется
      k0++;           //принимаем заявку в систему и
…
      for (i=0;i<N;i++)  //ищем свободный прибор
        if (s[i]==0)     //как только находим
          {s[i]=1;       //занимает прибор,
начинаем обслуживание и
          tau[i+1]=t+fvyb(); //моделируем
окончание обслуживания
          break; } }     //выходим из цикла поиска
и …
    }                  //конец sob0
```

Заметим, что состояния приборов первого узла определяются значениями `s[i]`, причем индекс изменяется от 0 до $N-1$, а моменты окончания обслуживаний в приборах определяются значениями `tau[j]` с индексами от 1 до N. Следовательно, прибору с номером j соответствуют переменные `s[j-1]` и `tau[j]`.

Функция sob1

Закончилось обслуживание в приборе первого узла. Номер этого прибора равен `ms`, а его состояние определяется переменной `s[n]`, где `n=ms-1`.

Для формирования состояния системы прежде всего необходимо переменной `tau[ms]` присвоить значение `BigC`, поскольку обслуживший заявку прибор либо станет свободным, либо останется занятым, пока обслуженная в нем заявка не покинет систему после обслуживания в приборе второго узла. В любом случае новый локальный процесс в нем не начнется.

После этого необходимо смоделировать наступление (или не наступление) события: «Заявка покидает систему после первого обслуживания», которое наступает с вероятностью p. Для этого необходимо обратиться к функции `bsv` и получить значение случайной величины, равномерно распределенной на интервале (0,1). Если это значение меньше p, то будем считать, что событие произошло. Заявка покидает систему, обслуживший ее прибор переходит в состояние ожидания. Если событие не наступает, обслуживший ее прибор

становится занятым. Заявка из него становится в очередь к прибору второго узла, а если до этого в очереди заявок не было и прибор, следовательно, был свободен, то начинается обслуживание и необходимо смоделировать момент окончания обслуживания. Таким образом, функцию можно записать так.

```
   void sob1(void) {       //закончилось обслуживание
в первом узле
     int n=ms-1;           //индекс прибора,
обслужившего заявку
     tau[ms]=BigC;
     if(bsv()<p)           //проверяем условие ухода
заявки
       {                 //если условие выполняется
       k0--;             //заявка покидает систему
       s[n]=0; }           //прибор становится
свободным
     else                  //если условие не
выполняется
       {k2++;            //заявка поступает во второй
узел
       if(k2==1)           //если прибор был свободен
         tau[N+1]=t+fobs();    //моделируем
окончание обслуживания
       s[n]=2; }           //прибор первого узла
остается занятым
     }                   //конец sob1
```

Функция sob2

Окончания обслуживания в приборе второго узла приводит к уменьшению на единицу количество заявок в системе, в очереди ко второму прибору и к освобождению одного из занятых приборов первого узла. Если в очереди к прибору второго узла остаются заявки, надо взять очередную на обслуживание, смоделировать момент окончания этого обслуживания и присвоить переменной `tau[N+1]`. Если же заявок в очереди не осталось, то переменной `tau[N+1]`

следует присвоить значение BigC. Следовательно, текст функции может быть записан в виде:

```
   void sob2(void) {       //закончилось обслуживание
во втором узле
     k0--;   k2--;             //заявка уходит из
системы и из очереди
     for (int i=0;i<N;i++)//начинаем поиск занятого
прибора узла 1
       if (s[i]==2)        //находим первый из
занятых,
         {s[i]==0; break;}//освобождаем его и
выходим из поиска
     if (k2>0)             //если в очереди остались
заявки, то
       tau[N+1]=t+fobs(); //начинаем новое
обслуживание иначе
     else tau[N+1]=BigC;  //переводим прибор в
состояние ожидания
     }                  //конец sob2
```

При правильном выборе параметров СМО, законов $F_{вх}$, $F_{выб}$ и $F_{обс}$ и соответствующих им алгоритмов моделирования на модели второй версии можно выполнить более точное исследование процесса функционирования магазина. Но и эта модель, естественно, не совершенна. Она, например, не учитывает поступление товаров в магазин от оптовых поставщиков. Учтем это в следующей версии.

СМО «Минимаркет». Версия 3

Для моделирования движения товаров сделаем следующие предположения. Будем предполагать, что в магазине продаются товары M наименований. Соответственно, в СМО определим M типов заявок, которые будут моделировать единицы каждого товара. Заявки этих типов далее будем называть просто «товар». Пусть эти заявки не имеют никаких параметров, т. е. при моделировании будем учитывать только тип товара и не будем принимать во внимание другие свойства и особенности как, например, вес, цена или объем.

Для моделирования запасов товаров в магазине определим в СМО M накопителей для товаров. Будем предполагать, что в каждом

накопителе могут находиться товары только одного типа и количество этих заявок в каждом накопителе не должно превышать некоторого значения L. Для более точного моделирования надо бы предположить, что объемы накопителей разные, но это бы только усложнило код программы моделирования и ничего не дало для понимания логики ее написания.

Будем предполагать, что неограниченные запасы заявок-товаров находятся в приборе «склад», который будет моделировать поставщиков товаров в магазин. Работа этого прибора состоит в формировании заявки типа «доставка», которая поступает на обслуживание в прибор второго узла и моделирует партию товаров, доставляемую поставщиком в магазин. Доставка имеет M параметров, которые определяют, сколько товаров надо добавить в накопители после ее обслуживания в приборе второго узла.

Пусть на прибор второго узла поступает поток G, события которого моделируют моменты времени, определенные графиком $G_{зв}$, когда продавец должен позвонить поставщику и заказать доставку товаров в магазин и когда товары будут доставлены в магазин.

Схема СМО третьей версии представлена на рис. 4.3. Как и в предыдущей версии заявки, моделирующие покупателей, поступают в систему одним потоком и обслуживаются в ней по такой же дисциплине, как и в СМО второй версии. Заявки этого типа (далее «покупатели») принимаются в СМО, если есть свободный прибор в первом узле. После обслуживания в первом узле покупатель либо с вероятностью p покидают систему, освобождая прибор, либо с вероятностью $1–p$ поступает во второй узел, оставляя прибор занятым.

Рис. 4.3. СМО «Минимаркет-3»

Обслуживание каждой заявки покупателя в приборе второго узла состоит в том, что из накопителей удаляется некоторые случайные количества товаров, которые вместе с покупателем покидают систему. Эти количества удаляемых товаров определяются по результатам моделирования. Будем предполагать, что заданы вероятности того, что покупатель пожелает купить за одно посещение j единиц i-го товара. Если в результате моделирования окажется, что с заявкой покупателем должны покинуть систему больше заявок-товаров некоторого типа, чем их есть в соответствующем накопителе, то из этого накопителя удаляются все заявки. Длительность обслуживания, определяемая

законом $F_{обс}$, может и должна зависеть от количества выбранных покупателем единиц каждого товара.

Как только наступает событие в потоке *G*, прибор второго узла дообслуживает очередного покупателя и начинает обслуживать наступившее событие в течение времени, которое определяется законом $F_{зв}$. Этот вид обслуживания моделирует заказ доставки товаров поставщиком. Закончив обслуживание события, прибор вновь обслуживает покупателей, а прибор склад с этого момента начинает работу. Длительность его работы определяется законом $F_{скл}$. Работа склада моделирует интервал времени до момента доставки поставщиком партии товаров в магазин. Закончив работу, склад направляет в прибор второго узла заявку-доставку и переходит в состояние ожидания до тех пор, пока в приборе второго узла не обработается следующее событие из потока *G*. Будем полагать, что параметры заявки-доставки равны стольким количествам товаров, сколько их необходимо для пополнения накопителей до максимального количества *L* на момент обслуживания прибором события из потока *G*.

После обслуживания очередного покупателя прибор второго узла начинает обслуживать событие из потока *G*. Если событие в потоке *G* не наступило к этому моменту, то прибор берет на обслуживание доставку. Длительность обслуживание доставки определяется законом F_{dos}. Если ни событие из потока *G*, ни заявка-доставка не ожидают обслуживания, прибор берет на обслуживание очередного покупателя, если он есть в очереди. Если и покупателя нет, то прибор переходит в состояние ожидания.

Описание программы моделирования

Параметры

Введем для параметров системы следующие константы и переменные:

```
#define N 100 //максимальное число заявок-покупателей системе
#define M 100 //количество типов (наименований) товаров
#define L 60  //объемы накопителей для заявок-товаров
float p;      //вероятность ухода заявки-покупателя из системы
              //после первого обслуживания
```

```
   float q[M][L]; // q[i][j]–вероятности того, что
покупатель пожелает
                  //приобрети в магазине не больше j
            единиц i-го товара
                  //q[i][0]–вероятность не желания
            преобретать товар i
                  //q[i][L-1]=1 для всех i
```

Функции моделирования локальных процессов

Будем предполагать, что для моделирования длительностей локальных процессов в СМО третьей версии определены функции:

```
   double fvx(void);  //ожидание заявки-покупателя
   double fvyb(void); //обслуживание в первом узле
   double fobs(void); //обслуживание покупателя во
втором узле
   double fdos(void); //обслуживание доставки во
втором узле
   double fzv(void);  //обслуживание события из
потока G
   double fskl(void); //длительность работы прибора
склад
   double fg(void);   //интервал до следующего
события в потоке G
```

Вектор состояния

В вектор событий этой версии введем все переменные состояния, что были определены во второй версии.

```
   int k0;        //количества покупателей в
системе,
   int s[N]; //состояния приборов первого узла: 0—
свободен,
                  //1—обслуживает, 2—занят пока
покупартель в СМО
   int k2;        //количество покупателей в очереди
ко второму прибору
```

Эти переменные полностью определяют состояние первого узла и очередь покупателей к прибору второго узла. Но если переменная k2 в предыдущей версии определяла и состояние прибора второго узла, то в этой версии это не так. Во втором приборе три типа обслуживаний и это надо отразить в векторе состояния. Для этого введем переменную

```
int s2;          //состояние прибора второго узла
```

Значения этой переменной будут означать следующие: 0–прибор свободен; 1 – обслуживает покупателя; 2 – обслуживает доставку (магазин принимает товар); 3 – обслуживает событие из потока G (продавец магазина по телефону заказывает товар).

Состояния второго узла определяется также количествами товаров в накопителях на текущий момент модельного времени, наступило или нет к данному моменту событие в потоке *G*, ожидает или нет обслуживания заявка-доставка, а если ожидает, то сколько при ее обслуживании необходимо добавит товаров в накопители. Поэтому введем следующие переменные в вектор состояния:

```
int kt[M];//запасы товаров в накопителях
int zd;          //индикатор доставки: 1—ожидает обслуживания; 0—её нет
int ktzd[M];     //количества товаров в доставке
int tzv;         //индикатор события из G: 1—наступило, 0—нет
```

И, наконец, введем перемену, которая будет определять состояние склада.

```
int sc;          //состояние склада: 0—свободен; 1—готовит доставку
```

Вектор событий

В СМО третьей версии, как и в предыдущей, локальные процессы ожидания покупателей и обслуживания их в первом узле образуют *N*+1 последовательность локальных процессов. Во втором узле все три типа обслуживаний в приборе образуют одну последовательность локальных процессов. Но кроме этого поток событий *G* образует отдельную последовательность локальных процессов ожидания событий, и обслуживания в приборе склад образуют другую последовательность. Следовательно, количество *K* компонент вектора событий равно *N*+4.

```
   #define K 104       //Количество потоков основных
событий
   double tau[K];      //вектор событий
```

Выберем отличное от второй версии назначение компонент вектора. Пусть `tau[0]` – момент поступления очередного покупателя; `tau[1]` – момент окончания обслуживания в приборе второго узла; `tau[2]` – момент наступления очередного события в потоке G; `tau[3]` – момент окончания работы прибора склад; `tau[4],…,tau[3+N]` – моменты окончаний обслуживаний в приборах первого узла.

Блок «Модель»

Блок «Модель» в главной функции программы запишем в виде.

```
   switch(ms) {            //обработка событий
     case 0: sob0(); break;    //приход покупателя
     case 1: sob2(); break;    //окончание
обслуживание в приборе узла 2
     case 2: sobG(); break;    //событие в потоке G
     case 3: sobS(); break;    //окончание
обслуживания в приборе склад
     default: sob1(); }        //окончание
обслуживания в первом узле
```

Функция sob0

Поскольку в этой СМО третьей версии дисциплина поступления заявки покупателя в систему и обслуживания в первом узле точно такие же, как и в предыдущей версии, то и функция обработки факта появления покупателя на входе системы должна быть такая же. Но в этой версии вектор событий иной. Прибору с номером j (j = 1, 2, …, N) соответствуют переменные `s[j-1]` и `tau[j+3]`. С учетом этого функция запишется в виде.

```
   void sob0(void)         //поступила заявка на вход
системы
     {int i;           //рабочая переменная
     tau[0]=t+fvx();       //момент прихода следующей
заявки
```

```
    if (k0<N) {           //если условие приема
выполняется
      k0++;           //принимаем заявку в систему и
…
      for (i=0;i<N;i++)  //ищем свободный прибор
        if (s[i]==0)     //как только находим
          {s[i]=1;       //занимает прибор,
начинаем обслуживание и
          tau[i+4]=t+fvyb(); //моделируем
окончание обслуживания
          break; } }     //выходим из цикла поиска
и …
    }                 //конец sob0
```

Функция sob1

При окончании обслуживания заявки в приборе первого узла необходимо вычислить индекс прибора, т. е. индекс компоненты массива s, которая определяет состояние прибора, а также присвоить `tau[ms]` значение `BigC`, поскольку другое обслуживание в приборе не начнется. Затем необходимо проверить условие ухода покупателя из системы, а для этого получить значение случайной величины, равномерно распределенной в интервале (0,1). Если условие выполняется, то необходимо уменьшить количество заявок в системе и соответствующей прибору компоненте массива s присвоить значение 0, т. е. освободить прибор. Если условие не выполняется, то покупатель должен встать в очередь к прибору второго узла. Если при этом прибор свободен, то должно начаться обслуживание поступившего в очередь покупателя.

```
  void sob1(void) {       //закончилось обслуживание
в первом узле
    int n=ms-4;           //индекс прибора,
обслужившего заявку
    tau[ms]=BigC;
    if(bsv()<p)           //проверяем условие
наступления события
      {                   //если условие выполняется
      k0--;               //заявка покидает систему
```

```
      s[n]=0; }          //прибор становится
свободным
    else                 //если событие не
наступает
      {k2++;          //заявка поступает во второй
узел
      if(s2==0)          //если прибор был свободен
        {s2=1;           //заявка принимается на
обслуживание
        tau[1]=t+fobs(); //моделируем окончание
обслуживания
      s[n]=2; }          //прибор первого узла
остается занятым
    }                 //конец sob1
```

Функция sob2

Обработка окончания обслуживания в приборе второго узла зависит от того, какое из трех видов обслуживаний закончилось, т. е. от состояния прибора, от значения переменной `s2`. Поэтому первым оператором в функции может быть оператор

```
    switch (s2) {        //окончание
обслуживания...
      case 1: zout(); break; //...покупателя
      case 2: prt();  break; //...приемки товара
      case 3: zv(); }        //...звонка на склад
```

Здесь функции zout, prt и zv выполняют все действия по формированию состояния системы после окончания обслуживания, кроме формирования состояния самого прибора, поскольку следующее его состояние не зависит от того, какое обслуживание завершилось. А новое состояние прибора зависит от того, что ожидает обслуживания. По дисциплине обслуживания, прежде всего, необходимо проверить, ожидает ли обслуживания событие из потока G. Если нет, то дожидается ли обслуживания доставка. Если и доставка не поступила к этому моменту, то на обслуживание можно взять покупателя. В соответствии с этим надо задать значения переменным `s2` и `tau[1]`. Таким образом, в окончательном виде функция может быть следующей.

```
  void sob2(void) {       //окончание обслуживания в
приборе узла 2
    switch (s2) {         //окончание
обслуживания...
      case 1: zout(); break; //...покупателя
      case 2: prt();  break; //...приемки товара
      case 3: zv(); }        //...звонка на склад
        //формируем состояние прибора второго узла
    if (tzv==1)           //если ждет событие из
потока G
      {s2=3;              //событие принимаем на
обслуживание
      tzv=0;          //изменяем индекс события
      tau[1]=t+fzv(); }  //моделируем длительность
обслуживания
    else                  //если событие из G еще не
наступило, то
      if (zd==1)          //если ждет заявка-
доставка
        {s2=2;            //принимаем ее на
обслуживание
        zd=0;             //изменяем индекс заявки-
доставки
        tau[1]=t+fdos(); }    //моделируем
длительность обслуживания
      else                //если и заявки-доставки
нет, то
        if (k2>0)         //если ждет обслуживания
покупатель
        {s2=1;            //начинаем обслуживать
покупателя
        tau[1]=t+fobs(); }    //моделируем
длительность обслуживания
```

```
        else          //если и покупателя нет в
очереди
           { s2=0;          //прибор переходит в
состояние ожидания
           tau[1]=DigC; } //окончание обслуживания
неопределено
     }                  //конец функции sob2
```

Функция zout предназначена для формирования состояния системы после обслуживания в приборе второго узла очередного покупателя. Прежде всего, необходимо уменьшить на единицу значения переменных k2 и k0, поскольку после обслуживания покупатель должен покинуть систему. Затем необходимо смоделировать количество товаров, которые должны покинуть систему вместе с покупателем. И, наконец, удалить из накопителей, т. е. вычесть из элементов массива kt, число фактически покидающих систему товаров либо обнулить соответствующие элементы.

```
   void zout(void)          //окончание обслуживания
покупателя
     {int i,j;              //рабочие переменные
     float m;
     k2--;  k0--;           //заявка покидает очередь
и систему
              //определяем количества товаров,
которые унесет покупатель
     for (i=0;i<M;i++)          //цикл по типам
товаров
       if (kt[i]>0)         //если товара типа i есть
в запасах
         {m=bsv();          //получаем значение
случайной величины
         for(j=0;j<L;j++) //цикл по количеству
единиц товара
           if (m<q[i][j]) {    //если покупатель
желает взять j единиц
```

```
            if (kt[i]<j) //если запасов не хватает
              kt[i]=0;       //покупатель уносит
все запасы
            else           //если запасов хватает
              kt[i]=kt[i]-j; //уменьшаем запасы
товара на j единиц
            break; }         //выход из цикла по
количеству товара
         }                 //конец выбора количества
товара i
     }                  //конец функции zout
```

Функция prt выполняется после окончания обслуживания доставки. В ней необходимо изменить индекс доставки на 0 и пополнить запасы, т. е. к элементам массива kt добавить элементы массива ktzd.

```
   void prt(void)          //окончание приемки
товаров со склада
     { int i;
     zd=0;                 //обнуляем индекс доставки
     for (i=0;i<M;i++)
       kt[i]=kt[i]+ktzd[i];    //пополняем запасы
                  }                //конец функции
prt
```

Функция zv также достаточно проста. Необходимо сформировать доставку, т. е. вычислить, сколько на текущий момент времени в запасах недостает каждого товара до максимального уровня и занести эти значения в массив ktzd. Затем необходимо запустить в работу прибор склад, вычислить момент окончания его работы, и присвоить ноль индикатору события.

```
   void zv(void)           //окончание звонка на
склад
     { int i;
     for (i=0;i<M;i++)
```

```
      ktzd[i]=L-kt[i];        //формируем заявку-
доставку
    sc=1;                   //начинает работу склад
    tau[N+3]=t+fskl();        //время окончания
работы склада
    tzv=0;                  //событие обслужено
    }                     //конец функции zv
```

Функция sobG

Эта функция вызывается в программе моделирования в момент наступления события в потоке *G* и должна выполнять всего 4 действия: присвоение индикатору события значения 1; моделирование момента наступления следующего события; если прибор второго узла свободен, то изменяем его состояние на обслуживание события; моделируем момент окончания этого обслуживания.

```
  void sobG(void)          //наступление события в
потоке G
    { tzv=1;            //событие наступило
    tau[ms]=t+fg();       //моделируем момент
следующего события
    if (s2==0)            //если прибор узла 2
свободен
      {s2=3;            //прибор начинает обслуживать
событие
      tau[1]=t+fzv();     }    //моделируем
окончание обслуживания
    }                     //конец sobG
```

Функция sobS

Окончание обслуживания в приборе склад приводит к тому, что появляется заявка-доставка и, если прибор второго узла свободен, начинается ее обслуживание, а склад переходит в состояние ожидания.

```
  void sobS(void)          //закончил работу склад
    { zd=1;             //появилась доставка
    sc=0; tau[ms]=BigC;  //склад освободился
```

```
    if (s2==0)              //если прибор узла 2
свободен
       {s2=2;           //начинает обслуживать
доставку
       tau[1]=t+fdos();    }    //время окончания
обслуживания
    }                   //конец sobS
```

Третья версия системы массового обслуживания еще боле полно позволяет исследовать процесс функционирования магазина. Но это и самая сложная модель. Сложен не только код программы моделирования, но и подготовка к проведению экспериментов. А это выбор значений параметров системы, выбор законов, определяющих длительности локальных процессов, выбор параметров этих законов, планирование экспериментов, обработка и представление результатов. Ввод исходных данных также представляет определенную сложность, поскольку в приведенных функциях обработки событий не предусмотрены проверки особых ситуаций, которые могут возникнуть при неправильном выборе или вводе значений параметров. Например, при неправильном выборе функций моделирования локальных процессов и их параметров события в потоке G могут появляться настолько часто, что прибор не будет успевать их обрабатывать. Для исключения таких особых ситуаций не обязательно проверять возможность их возникновения в процедурах обработки событий. Это необходимо делать в функциях ввода исходных данных и планирования эксперимента.

СМО «Мастерская». Версия 1

Предположим, что мастерская может одновременно выполнять N заказов, в мастерской работают M мастеров, один приемщик и заказы мастерская выполняет точно в срок. Определим в СМО один прибор первого узла обслуживания, который будет моделировать работу приемщика, и M однотипных приборов второго узла, которые будут моделировать работу мастеров. Длительности обслуживания во всех приборах определим как случайные величины с функцией распределения F_1 для первого прибора и F_2 для всех остальных приборов. Схема СМО изображена на рис. 4.5.

Рис. 4.5. СМО «Мастерская». Версия 1

Пусть на первый прибор поступает ординарный поток однотипных заявок без параметров, который будут моделировать клиентов. Длительности интервалов времени между поступлениями очередных заявок будем считать случайными величинами с функцией распределения F_0.

Определим следующую дисциплину обслуживания. Каждая поступающая заявка принимается в систему, если общее количество находящихся в СМО заявок не превышает N. Принятая заявка поступает в очередь к прибору первого узла. Если прибор свободен, заявка начинает обслуживаться, иначе заявка ожидает, пока не обслужатся в приборе впереди стоящие заявки. После обслуживания в первом узле заявка поступает во второй узел, а прибор первого узла берет на обслуживания следующую заявку из очереди или переходит в состояние ожидания, если очередь пуста.

Поступившая во второй прибор заявка становится в более короткую очередь к прибору второго узла. Если очереди равны или отсутствуют, заявка направляется к прибору с наименьшим номером. Все приборы обслуживают первую из стоящих в очереди заявку, и заявка остается в очереди до окончания обслуживания. Если заявок в очереди к прибору нет, то прибор пребывает в состоянии ожидания. После обслуживания во втором узле заявка покидает систему.

Описание программы моделирования

Параметры системы

Как и предыдущих примерах параметры СМО зададим следующим образом.

```
   #define N 100       //макимальное количество
заявок в системе
   #define M 5         //количество приборов в узла 2
```

Функции моделирования локальных процессов

Будем предполагать, что для моделирования длительностей локальных процессов определены соответствующие функции:

```
   double f0(void);    //длительность ожидание заявки
   double f1(void);    //длительности обслуживания в
прибore 1
```

```
double f2(void);    //длительности обслуживания в приборе 2
```

Вектор состояния

Состояние первого узла обслуживания полностью определяется количеством заявок в очереди к прибору, но только при условии, что обслуживаемая заявка остается в очереди до окончания ее обслуживания. Поэтому введем в вектор состояния системы переменную

```
int k1;             //число заявок в очереди к прибору узла 1
```

Состояние второго узла определяется состояниями его приборов, а их состояния, в свою очередь, количествами заявок в очередях к приборам. Введем в вектор состояния массив

```
int k2[M];      //число заявок в очередях к приборам узла 2
```

Значения компонент вместе со значением переменной `k1` будут полностью определять состояние системы.

Введем еще одну переменную в вектор состояния системы для значений общего количества заявок в системе

```
int k0;             //количество заявок в системе
```

Вектор событий

Входной поток заявок и процессы обслуживания в каждом приборе образуют отдельные последовательности локальных процессов, т. е. количество потоков основных событий $K=M+2$. Поскольку для определенности значение M выбрали равным 5, то K определим директивой

```
#define K 7         //количество потоков основных событий
double tau[K];      //вектор событий
```

Пусть `tau[0]` – момент поступления очередной заявки; `tau[1]` – момент окончания обслуживания заявки в приборе первого узла; `tau[2], ..., tau[M+1]` – моменты окончаний обслуживаний в приборах второго узла.

Блок «Модель»

Поскольку в СМО происходят локальные процессы только трех видов, то будет достаточно трех функций формирования состояний

системы после окончаний обслуживаний. Следовательно, блок «Модель» можно записать так:

```
   if (ms==0)       sob0();//поступила заявка
   else if (ms==1) sob1();//закончилось
обслуживание в узле 1
        else        sob2();//закончилось
обслуживание в узле 2
```

Функция sob0

Эта функция формирует состояния системы после появления на входе системы очередной заявки. Как и в предыдущих примерах, выполнение функции начинается с моделирования прихода следующей заявки и проверки условия приема ее в систему. Если заявка принимается в систему, то увеличиваем на единицу значения переменных k0 и k1 и проверяем занятость прибора первого узла. Если поступившая заявка оказалась единственной в очереди, то это значит, что прибор свободен и должен взять на обслуживание поступившую заявку.

```
   void sob0(void)          //на вход системы
поступила заявка
     {tau[0]=t+f0();        //моделируем время прихода
следующей
     if (k0<N){             //если заявка принимается
в систему
       k0++;           //увеличиваем количество
заявок в СМО
       k1++;           //увеличиваем очередь к
прибору 1
       if (k1==1)           //если заявка одна в
очереди
         tau[1]=t+f1();    }    //начинаем ее
обслуживание
     }                  //конец sob0
```

Функция sob1

Обработку события окончания обслуживания в приборе первого узла надо начинать с удаления заявки из очереди к прибору первого узла

и поиска самой короткой очереди во втором узле. Затем надо поставить заявку в найденную короткую очередь и начать ее обслуживание, если очередь была пуста. Эти действия можно записать в виде отдельной функции out1in2.

```
   void out1in2(void)          //переводим заявку из
узла 1 в 2
     {int i,j,k;              //рабочие переменные
     k1--;                    //удаляем заявку из
входной очереди
     for (k=N;i=M-1;i>=0;i--) //ищем короткую
очередь j в узле 2
       if (k2[i]<=k) {k=k2[i]; j=i;}
     k2[j]++;           //ставим заявку в очередь
     if (k2[j]==1)          //если заявка одна в
очереди
       tau[j+2]=t+f2();         //начинаем ее
обслуживание
     }                  //конец out1in2
```

Необходимо заметить, что цикл поиска короткой очереди начинается с больших значений индекса к меньшим, т. к. по дисциплине обслуживания заявка должна быть поставлена в самую короткую очередь с наименьшим номером.

В завершение обработки события окончания обслуживания в приборе первого узла надо сформировать состояние прибора после ухода заявки. Необходимо проверить, остались ли в первом узле заявки. Если заявки есть, надо смоделировать момент окончания обслуживания первой из них. Если заявок нет, то переменной tau[1] надо присвоить значение BigC. Окончательно функцию sob1 можно записать так

```
   void sob1(void){         //закончилось обслуживание
в узле 1
     out1in2();             // переводим заявку из
узла 1 в 2
         //формируем состояние узла 1 после
обслуживания заявки
```

```
     if (k1>0)              //если в очереди остались
еще заявки
       tau[1]=t+f1();       //начинаем обслуживать
первую из них
     else                   //если заявок в очереди
нет
       tau[1]=BigC;         //прибор узла 1 в
состоянии ожидания
     }                  //конец sob1
```

Функция sob2

Прежде чем формировать состояние системы после окончания обслуживания заявки в приборе второго узла необходимо вычислить индекс обслужившего заявку прибора и присвоить это значение некоторой переменной n. Затем надо уменьшить на 1 значения переменных k0 и k2[n] и сформировать состояние освободившегося от заявки прибора. Окончательно функцию можно записать в виде.

```
   void sob2(void)          //окончание обслуживания в
узле 2
     {int n=ms-2;           //индекс обслужившего
прибора
     k0--; k2[n]--;         //удаляем заявку из
очереди и системы
     if (k2[n]>0)           //если в очереди остались
заявки
       tau[ms]=t+f2();          //обслуживаем одну из
них
     else                   //если заявок в очереди
нет
       tau[n]=BigC;         //окончание обслуживания
не определено
     }                  //конец sob2
```

Данная модель позволяет только в общих чертах рассмотреть процесс функционирования мастерской. Усложним модель и в следующей версии учтем, например, выдачу клиентам их отремонтированных приборов.

СМО «Мастерская». Версия 2

Добавим к модели первой версии еще один третий узел обслуживания с N приборами, которые будут моделировать ожидания выполнения заказов и приход клиентов за своими отремонтированными приборами. Схема СМО изображена на рис. 4.6.

Будем предполагать, что в первом приборе выполняются три вида обслуживания, длительности которых определяются законами F_{11}, F_{12} и F_{13}. Первое или первичное обслуживание принятой в систему заявки моделирует осмотр и проверку приемщиком принесенного заказчиком прибора. Второе обслуживание моделирует оформление заказа и оплату ремонта. Третье обслуживание, это обслуживание заявки, поступившей из третьего узла. Оно моделирует выдачу заказчику отремонтированного прибора.

Как и в прежней версии M приборов второго узла будут моделировать работу мастеров. Не будем учитывать квалификацию и профессиональные навыки мастеров и предположим, что длительность обслуживания в каждом из этих приборов является случайной величиной с функцией распределения F_2.

Рис. 4.6. СМО «Мастерская». Версия 2

Длительность обслуживания в остальных N приборах состоит из постоянной составляющей, которая вычисляется по закону F_{31}, и случайной, длительность которой определяется функцией распределения F_{32}. Первая составляющая моделирует интервал времени, после которого можно получить отремонтированную вещь. Этот интервал назначает приемщик при оформлении заказа и в общем случае может зависеть от количества и занятости мастеров. Но заказчик, как правило, приходит позже назначенного срока исполнения заказа. Эта задержка и моделируется случайной составляющей длительности обслуживания.

Пусть как и в предыдущей версии, заявки поступают в СМО одним ординарным потоком и длительность интервалов между поступлениями очередных заявок определяется законом F_0. Если в системе уже есть N заявок, то поступающая заявка не принимается в систему. Если же заявка принимается в систему, за ней резервируется один из свободных приборов третьего узла.

Каждая принятая в систему заявка в порядке очереди проходит первый этап обслуживания в первом приборе. После этого заявка с вероятностью *p* покидает систему. Если заявка остается в системе, то она проходит второй этап обслуживания и поступает во второй узел обслуживания. Одновременно начинается обслуживание в зарезервированном за этой заявкой приборе третьего узла, т. е. моделируется время возвращение клиента за отремонтированным прибором.

Обслуживание заявок во втором узле выполняется по дисциплине, описанной в первой версии модели.

После окончания обслуживания в третьем узле заявка становится в отдельную, вторую очередь к первому прибору. Это очередь клиентов, которые пришли за своими приборами. Будем считать, что заявки имеют относительный приоритет перед заявками первой очереди, т. е. будем исходить из предположения, приемщик в первую очередь обслуживает клиентов, которые пришли забрать свои отремонтированные приборы.

Описание программы моделирования

Параметры

Параметры СМО определим следующим образом

```
#define N 100  //макимальное количество заявок в системе
#define M 5    //количество приборов в узле 2
float p;       //вероятность ухода заявки из системы после первого
               //вида обслуживания в приборе первого узла
```

Функции моделирования локальных процессов

Будем предполагать, что для моделирования длительностей локальных процессов определены функции

```
double f0(void)    //ожидание заявки
      //длительность обслуживания ...
double f11(void);  //первого вида в приборе 1
double f12(void);  //второго вида в приборе 1
double f13(void)   //в прибре 1 заявки из третьего узла
```

```
double f2(void);   //в приборе узла 2
double f3(void);   //в приборе узла 3
```

Вектор состояния

Для описания состояния первого узла этой версии системы уже недостаточно знать количество заявок в системе и в очередях. Надо иметь информацию о приборах третьего узла, о состоянии очереди заявок, поступающих из третьего узла в первый, и о состоянии прибора первого узла, т. е. какую заявку, из какой очереди он обслуживает. Поэтому будем описывать состояние первого узла следующими переменными:

```
int k0;        //общее количество заявок в системе
int k10;       //количество заявок во входной очереди
int k13;       //количество заявок в очереди 2 (от узла 3)
int r[N]; //r[i] – номер прибора, из которого поступила
               //заявка на i-е место в очередь 3, если место
               //свободно, значение равно 1
int s1;        //состояние прибора первого узла: 0-свободен;
               //1-первое и 2-второе обсуживание из входной очереди;
               //3-обслуживание заявки из узла 3
```

Состояние третьего узла можно описать значениями элементов массива

```
int s3[N];//состояния приборов третьего узла: 0—свободен
               //1—обслуживает заявку; 2—заявка уже в узле 1
```

Состояние второго узла определяется как и в первой версии значениями элементов массива *k*2.

```
int s2[M];//состояния приборов узла 2: 0–
свободен 1–обслуживает
```

Вектор событий

Входной поток заявок и события окончаний обслуживаний в приборах образуют отдельные потоки основных событий. Таким образом, количество потоков *K* равно *N*+*M*+2.

```
#define K 107       //количество потоков основных
событий
double tau[K];      //вектор событий
```

Пусть в векторе событий компонента имеют следующие значения: `tau[0]` – момент прихода очередной заявки; `tau[0]` – момент окончания очередного вида обслуживания в приборе первого узла; `tau[2]`, ..., `tau[M+1]` – окончания обслуживаний в приборах второго узла; `tau[M+2]`, ..., `tau[N+M+1]` – окончания обслуживаний в приборах третьего узла.

Блок «Модель»

Поскольку события окончания обслуживаний во всех приборах второго узла однотипные и в приборах третьего узла однотипные, то для формирования состояний системы и коррекции вектора событий достаточно четырех функций.

```
double sob0(void); //приход очередной заявки
double sob1(void); //окончание обслуживания в
приборе узла 1
double sob2(void); //окончание обслуживания в
приборе узла 2
double sob3(void); //окончание обслуживания в
приборе узла 3
```

С учетом введенных функций блок «Модель» в главной функции программы моделирования можно записать в виде.

```
if (ms==0)              sob0();
else if (ms==1)         sob1();
     else if (ms<M+3)   sob2();
          else          sob3();
```

Функция sob0

Эта функция отличается от функции sob0 в первой версии модели только обозначением компонент вектора состояния

```
   void sob0(void){        //на вход системы
поступила заявка
     tau[0]=t+f0();        //моделируем время прихода
следующей
     if (k0<N){            //если заявка принимается
в систему
       k0++;           //увеличиваем количество
заявок в СМО
       k10++;          //увеличиваем входную очередь
к прибору 1
       if (s1==0)          //если прибор узла 1
свободен
         {s1==1;           //начинаем первое
обслуживание и...
         tau[1]=t+f11();} }    //моделируем момент
его окончания
     }                   //конец sob0
```

Функция sob1

Как и прежде, в функции сначала будем описывать возможные изменения векторов состояния и событий, связанные с обслуженной заявкой, а потом сформируем состояние прибора первого узла.

Изменения состояния системы, которые произойдут после окончания обслуживания в приборе первого узла, зависят от того, какое именно обслуживание закончилось.

Если закончилось первичное обслуживание заявки, то необходимо смоделировать наступление события с вероятностью p. Если событие наступает, то заявка покидает систему, иначе заявка остается в приборе и начинается ее вторичное обслуживание, и других изменений в системе не произойдет. Если заявка покинет систему и надо сформировать новое состояние прибора первого узла.

Если закончилось вторичное обслуживание заявки из входной очереди, то заявка должна поступить во второй узел обслуживания. Связанные с этим действия, изменяющие вектора состояния и событий,

точно такие же, как и в первой версии модели. Необходимо найти свободный прибор в узле 3 и запустить его в работу. Опишем эти действия также в виде отдельной функции out1in2.

```
   void out1in2(void)          //переводим заявку из
узла 1 в 2
     {int i,j,k=N;             //рабочие переменные
     k11--;                  //удаляем заявку из
входной очереди
     for (i=M-1;i>=0;i--) //ищем короткую очередь j
в узле 2
       if (k2[i]<=k) {k=k2[i]; j=i;}
     k2[j]++;          //ставим заявку в очередь
     if (k2[j]==1)         //если заявка одна в
очереди
       tau[j+2]=t+f2();        //начинаем ее
обслуживание
     for (i=0;i<N;i++)         //ищем свободный
прибор узла 3
       if (s3[i]==0)       //как только находим
         {s3[i]=1;         //запускаем его в работу
         tau[M+3+i]=t+f3();    //вычисляем момент
окончания его работы
         break;}           //прекращаем поиск прибора
     }                   //конец out1in2
```

Если закончилось обслуживание заявки, поступившей из третьего узла, то заявка покидает систему и очередь, в которой она находилась. Еще при этом освободится один из занятых приборов третьего узла. И эти действия опишем в виде отдельной функции outS.

```
   void outS(void)           //удаляем заявку из
очереди и системы
     {k13--; k0--;           //сокращаем очередь и
общее число заявок
```

```
    for (i=0;i<N;i++)          //ищем занятый прибор
узла 3
      if (s3[i]==2)        //как только находим
        {s3[i]=0; break;}//освобождаем и
прекращаем поиск
    }                  //конец out1outS
```

Для формирования состояния первого прибора после его освобождения необходимо сначала проверить состояние очереди заявок, поступивших из третьего узла. Если в ней заявка есть, надо взять ее на обслуживание. Если заявок в этой очереди нет, то надо взять на обслуживание заявку из входной очереди. Если и входная очередь пуста, то прибор должен перейти в состояние ожидания.

Окончательно функцию sob1 можно записать в виде.

```
  void sob1(void)
    { double x; int i;         //рабочие переменные
    switch(s1) {
      case 1:              //закончилось первичное
обслуживание
        if (bsv()<p)       //заявка остается в
системе
          {s1=2;           //начинается вторичное
обслуживание
          tau[1]=t+f12();//вычисляем момент его
окончания
          return; }        //выход из функции
        else
          {k0--; k11--;} //заявка уходит из системы
и очереди
        break;
      case 2:              //закончилось вторичное
обслуживания
        out1in2(); break;//заявка переходит в узел
2
```

```
      case 3:              //закончилось обслуживание
из узла 3
         out1outS(); break;    //заявка покидает
систему
      }              //конец оператора switch
            //формируем состояние прибора первого
узла
    if (k13>0)           //если есть заявки в
очереди от узла 3
      {s1=2;         //занимаем прибор и вычисляем
момент
      tau[1]=t+f13();}       //окончания
обслуживания
    else                 //если нет заявок в
очереди от узла 3
      if (k11>0)         //если есть заявка во
входной очереди
        {s1=1;           //занимаем прибор и
вычисляем момент
        tau[1]=t+f11();} //окончания первичного
обслуживания
      else               //если и входная очередь
пуста
        {s1=0;           //прибор становится
свободным и
        tau[1]=BigC;}        //обслуживание в нем
не начинается
    }                //конец функции sob1
```

Функция sob2

После окончания обслуживания в приборе второго узла необходимо сначала вычислить индекс обслужившего заявку прибора. Затем удалить заявку из очереди к прибору и занять прибор обслуживанием очередной заявки, если она есть в очереди либо перевести прибор в свободное состояние.

```
  void sob2(void)          //окончание обслуживания в
узле 2
    {int n=ms-2;           //индекс обслужившего
прибора
    k2[n]--;             //удаляем заявку из очереди к
прибору
    if (k2[n]>0)           //если в очереди остались
заявки
      tau[ms]=t+f2();        //обслуживаем одну из
них
    else                   //если заявок в очереди
нет
      {s2[n]=0;            //прибор в свободное
состояние
      tau[ms]=BigC;}       //окончание обслуживания
не определено
    }                    //конец sob2
```

Функция sob3

Когда закончилось обслуживание в приборе третьего узла, заявка становится в очередь к первому прибору, а обслуживший прибор переходит в состояние 2. Если в этот момент первый прибор свободен, необходимо его занять.

```
  void sob3(void)          //окончание обслуживания в
узле 3
    {int n=ms-M-2;         //индекс обслужившего
прибора
    k3++;                  //заявку в очередь к
прибору узла 1
    if (s1==0)             //если прибор узла 1
свободен
      {s1=;              //занимаем прибор и вычисляем
момент
      tau[ms]=t+f2();}       //окончания
первичного обслуживания
```

```
    s3[n]=3;          //прибор узла 3 остается
занятым, но
    tau[ms]=BigC;          //не обслуживает заяву
    }                 //конец sob2
```

Итак, как видно из приведенных примеров, процедуры обработки событий по сути дела представляют собой запись на языке программирования дисциплины обслуживания моделируемой СМО с точки зрения происходящих в ней событий. Каждому типу основного события соответствует процедура, в которой сначала выполняются те действия по формированию векторов состояния и событий, которые обязательно происходят при наступлении данного события. Затем анализируются все возможные состояния системы, при которых может произойти данное событие и в зависимости от того, которое из них является текущим в данный момент, выполняются соответствующие изменения состояния и корректируются моменты наступления основных событий, т. е. изменяется вектор событий.

Список литературы

1. Шеннон Р. Имитационное моделирование систем – Искусство и наука. – М.: Мир, 1978. – 417 с.
2. Соболь И.М. Численные методы Монте-Карло. – М.: Наука, 1973. – 312 с.
3. Полляк Ю. Г. Вероятностное моделирование на электронных вычислительных машинах. – М.: Советское радио, 1971. – 400 с.

Оглавление

Учебное издание

ОСЛИН Борис Георгиевич

МОДЕЛИРОВАНИЕ
Имитационное моделирование СМО

Учебное пособие

Научный редактор
доктор технических наук,
профессор В. Г. Спицын

Редактор

Верстка

Отпечатано в Издательстве ТПУ в полном соответствии
С качеством предоставленного оригинал-макета

Подписано к печати. Формат 60x84/16. Бумага «Снегурочка».
Печать Xerox. Усл.печ.л. . Уч.-изд.л. .
Заказ . Тираж экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO
9001:2000

144

. 634050, г. Томск, пр. Ленина, 30.
Тел./факс: 8(3822)56-35-35, www.tpu.ru