электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.94

Реализация имитационной модели на основе многопоточного программирования

***Федоренко Ю.С.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
кафедра «Системы обработки информации и управления»

Научный руководитель: Черненький М.В.,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
chernen@bmstu.ru

1. Введение

Для моделирования технических, экономических, информационных процессов невозможно обойтись без вычислительной техники. При этом вычислительная сложность практических задач растет существенно быстрее, чем мощность процессоров. Вместе с тем многоядерные процессоры получают все большее распространение, поэтому для ускорения производительности разработчикам необходимо учитывать предоставляемые возможности и создавать программный код таким образом, чтобы он мог выполняться параллельно.

Использование многопоточности позволяет увеличить производительность следующим образом[1]:

1. ускорение выполнения приложений, реализуемое благодаря распараллеливанию потоков, выполняемых приложением, и их запуску на разных ядрах процессора;

2. благодаря многопоточности можно за то же время выполнить большее число вычислений по сравнению с однопоточным приложением;

3. многопоточность бывает полезна при наличии в приложении длительных операций, которые целесообразно вынести в отдельный поток.

Среди различных задач математического моделирования обычно в отдельную группу выделяют дискретное имитационное моделирование. Данный подход описывает процессы универсальным способом, опираясь на событийное представление независимо от сложности процесса, что представляется важным, поскольку лишь достаточно простые процессы удается промоделировать аналитическими методами.

При программной реализации задачи имитационного моделирования необходимо иметь возможность одновременно моделировать развитие многих процессов и их

взаимодействие. Для решения этой проблемы существует общепринятый подход на основе списков текущих и будущих событий. В статье предлагается подход на основе многопоточности и излагаются его преимущества и особенности.

2. Постановка задачи

В качестве примера реализации предлагаемого подхода была выбрана классическая имитационная модель парикмахерской, имеющая следующие функциональные параметры:

1. Поддержка двух типов клиентов (мужчины и женщины), причем задается интервал времени появления для каждого из типов клиентов.

2. Наличие двух обслуживающих аппаратов (мастеров). В настройках модели задается время обслуживания; клиент становится в очередь к тому мастеру, где меньше претендентов. Если в очереди более определенного числа клиентов (что также задается в настройках), то клиент не ждет обслуживания, а покидает парикмахерскую.

3. По окончанию обслуживания каждый клиент направляется в общий обслуживающий аппарат (кассу) для оплаты оказанных услуг. Время обслуживания на кассе также задается индивидуально.

4. Регулируется общее время длительности моделирования. Задается реальное время (в секундах), как интервал, который необходимо промоделировать; также задается время, в течение которого будет производиться моделирование.

5. По окончанию работы программы собирается статистика о числе обслуженных клиентов, о среднем времени ожидания в очереди, о временах обслуживания.

6. Обеспечивается графическая визуализация происходящего процесса моделирования.

3. Описание модели на псевдоязыке описания сцепленных процессов

3.1. Блочная схема модели

В данной модели используются блоки-контроллеры и блоки-процессоры [2]. Блок-процессор содержит трек объединенных операторов. Он не содержит инициаторов, но их множество может входить извне и обрабатываться параллельно. Характерной особенностью блока-процессора является взаимодействие через локальные среды. Блок-контроллер представляет собой специальным образом организованный агрегат. Он предназначен для формирования инициаторов для других блоков.

На рисунке 1 представлена блочная схема модели. Блок «Распределитель» обеспечивает распределение инициаторов по блокам «Мастер1» и «Мастер2». Это

делается на основе анализа параметров «Длина_очереди_1» и «Длина_очереди_2». Инициатор направляется к тому блоку «мастер», длина очереди к которому меньше.

Алгоритмы процесса обслуживания в блоках «Мастер1» и «Мастер2» структурно одинаковы и различаются лишь значениями параметров C1 и D1.

Алгоритмы процесса создания инициаторов в блоках «Генератор1» и «Генератор2» структурно одинаковы и различаются лишь значениями параметров A1, B1, C1 и D1.

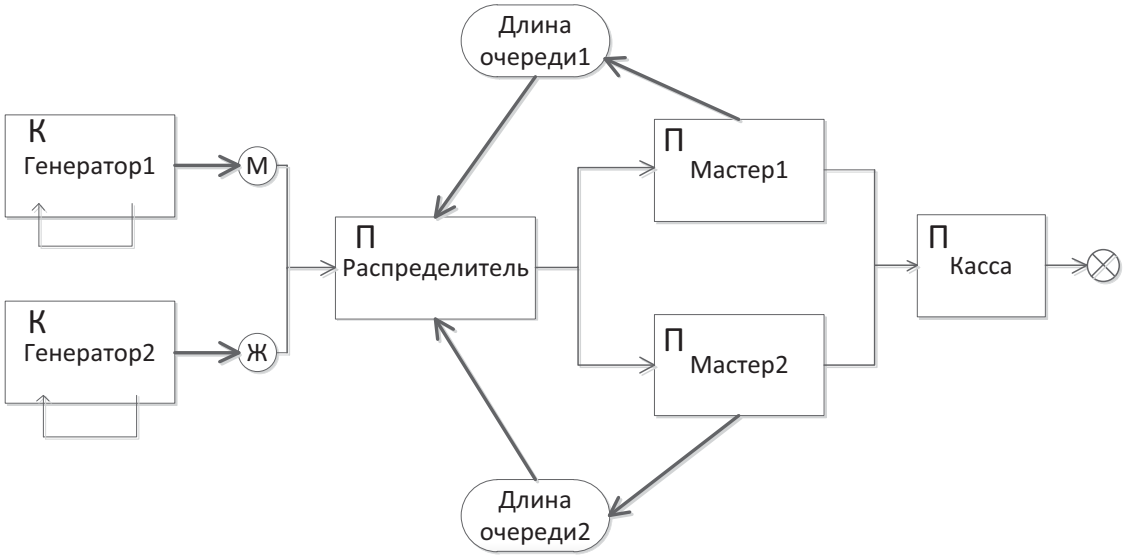

Рис. 1. Блочная схема модели обслуживания

Блок-контроллер Генератор1

Описание

A1, B1, C1, D1, Тм – скаляры; НАЧ - метка;

Все описание;

Алгоритм

НАЧ: создать ЛС типа вектор (1-2 – скаляры);

Создать АВ типа ссылка;

ЛС(1) := A1 + RAND (B1 – A1);

ЛС(2) := ВРЕМЯ;

АВ := ссылка на вектор ЛС;

Активизировать инициатор из АВ в блок РАСПРЕДЕЛИТЕЛЬ на метку ВХОД;

Тм := ВРЕМЯ + (C1 + RAND (D1 – C1));

Ждать ВРЕМЯ = Тм;

Направить ИНИЦИАТОР на метку НАЧ;

Всё алгоритм;

Всё блок.

Блок-процессор Распределитель

Описание

Длина_очереди_1 – скаляр в блоке МАСТЕР 1;

Длина_очереди_2 – скаляр в блоке МАСТЕР 2;

ВХОД – метка;

Всё описание;

Алгоритм

ВХОД: Если (Длина_очереди_1 < Длина_очереди_2),

То направить *ИНИЦИАТОР* в блок МАСТЕР1 на метку НАЧАЛО,

Иначе в блок МАСТЕР2 на метку НАЧАЛО;

Всё алгоритм.

Блок-процессор Мастер1

Описание

Длина_очереди_1 – скаляр; // начальное значение 0

Мастер1 – скаляр; // начальное значение «свободен»

НАЧАЛО – метка;

C1, D1, Тоб – скаляр;

Все описание;

Алгоритм

НАЧАЛО: Длина_очереди_1 := Длина_очереди_1 + 1;

Ждать Мастер1 = «свободен»;

Мастер1 := «занят»;

Тоб := ВРЕМЯ + (C1 + RAND (D1 – C1));

Ждать ВРЕМЯ = Тоб;

Мастер1:= «свободен»;

Длина_очереди_1 := Длина_очереди_1 – 1;

направить ИНИЦИАТОР в блок КАССА на метку НАЧАЛО;

Всё алгоритм;

Всё блок.

4. Описание используемых средств языка программирования Java

В контексте языка программирования Java процесс — это совокупность кода и

данных, разделяющих общее виртуальное адресное пространство. Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен, и взаимодействие между процессами осуществляется с помощью специальных средств.

Для каждого процесса среда выполнения создает так называемое «виртуальное адресное пространство», к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. Среда выполнения отвечает за то, как виртуальное пространство процесса проецируется на физическую память [3].

При запуске программы среда выполнения создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса. В среде Java один поток – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса. Однако следует помнить, что на одно ядро процессора в каждый момент времени приходится одна единица исполнения. Таким образом, одноядерный процессор может обрабатывать команды только последовательно. Однако запуск нескольких параллельных потоков возможен и в системах с одноядерными процессорами. В этом случае среда выполнения будет периодически переключаться между потоками, поочередно давая выполняться то одному, то другому потоку. Такая схема называется псевдо-параллелизмом. Система запоминает состояние (контекст) каждого потока, перед тем как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока [3]. Таким образом, при псевдопараллельном выполнении потоков процессор «мечется» между выполнением нескольких потоков, выполняя по очереди части каждого из них.

Схема взаимодействия потоков представлена на рисунке 2.

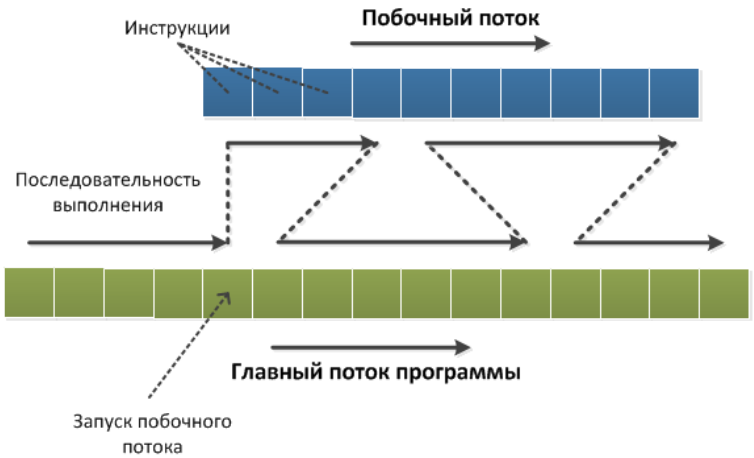

Рис. 2. Взаимодействие потоков

Цветные квадраты на рисунке – это инструкции процессора (зеленые – инструкции главного потока, синие – побочного). Выполнение идет слева направо. После запуска побочного потока его инструкции начинают выполняться вперемешку с инструкциями главного потока. Количество выполняемых инструкций за каждый подход не определено. Это обстоятельство может привести к конфликту, связанному с попыткой одновременного обращения к данным. Для регулирования этого явления в языке Java предусмотрены некоторые встроенные средства. При реализации данной задачи имитационного моделирования использовались следующие возможности Java:

1. Создание нового потока. В программе был создан специальный класс Transact, который реализовывает интерфейс *Runnable*, перегружая метод *run()*. Таким образом, для генерирования транзакта необходимо создать объект данного класса и вызвать у него метод *run( )*. В результате появится транзакт, запущенный в отдельном потоке.

2. Язык Java предоставляет несколько средств для работы с потоками [4]. Для приостановки потока используется метод *Thread.Sleep()*. В качестве параметра вызова задается количество миллисекунд, на которое поток будет остановлен.

При работе приложения к некоторым данным может быть произведено обращение из нескольких потоков. Для предотвращения подобных проблем, такие данные были реализованы в виде атомарных переменных (класс *AtomicInteger* и пр.). Таким образом, средствами языка гарантируется, что одновременное обращение к этим переменным не будет осуществлено более чем из одного потока.

Также для организации очередей использовался блок *syncronized*. Язык гарантирует, что данный блок все потоки проходят последовательно, таким образом, на его входе сформируется очередь.

Дополнительно для синхронизации взаимодействия между потоками используются методы *wait()* и *notify()*. Данных средств языка оказывается достаточно для реализации поставленной задачи имитационного моделирования.

5. Модель движения транзакта

В каждом из алгоритмических блоков (рисунок 1) инициатор движется определенным образом. Поэтому можно составить алгоритм движения инициатора [2]. Наиболее интересный алгоритм в блоке МАСТЕР приведен на рисунке 3.

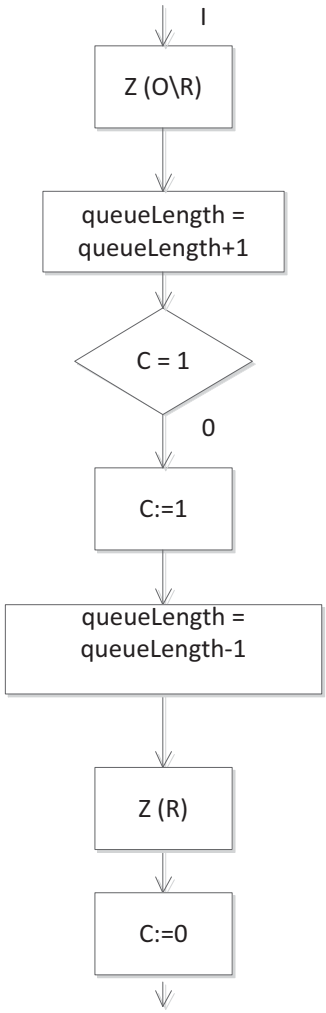

Рис. 3. Блок-схема движения транзакта в блоках МАСТЕР

На рисунке 3 видно, что необходимо реализовать очередь обслуживания и захват ресурса, т. е. пока обслуживающий аппарат занят обслуживанием одного инициатора, в блок не может войти следующий инициатор.

6.1. Реализация задачи моделирования с использованием многопоточного программирования

Логическое управление движением инициатора осуществляется путем введения элементарных операторов. При этом вводится h_i^y — условие продвижения инициатора по треку к соседним операторам по треку, h_i^t — условие продвижения по времени (задержка); $h_i^л$ — условие продвижения при истинном значении логической функции. Все эти условия продвижения инициаторов могут быть реализованы средствами многопоточности.

Каждый клиент, пришедший на обслуживание, моделируется в виде транзакта. Программно он представляется в виде отдельно запущенного потока. В коде это достигается путем использования отдельного класса, реализующего интерфейс Runnable. При генерации транзакта создается объект класса Transact, а затем для созданного объекта вызывается метод *run()*. При этом происходит создание нового потока для данного транзакта.

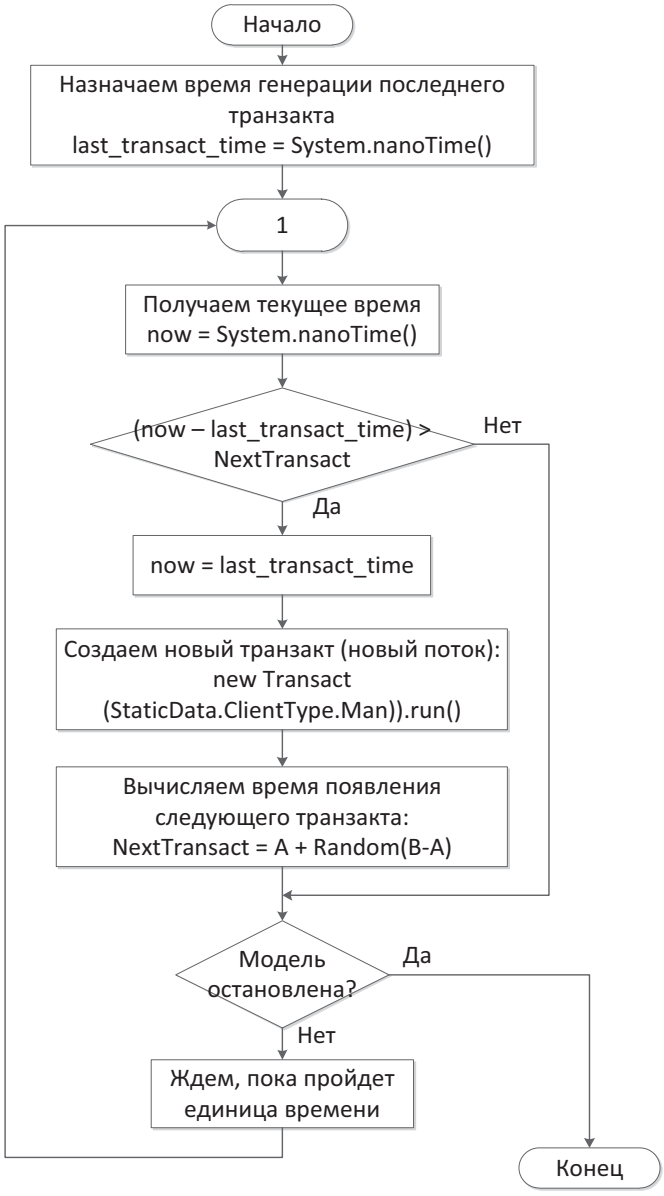

Рис. 4. Алгоритм реализации блока генератор

Алгоритм реализации генератора на языке Java представлен на рисунке 4. Класс генератор наследует (*extends*) от класса AnimationTimer (*javafx.animation.AnimationTimer*) и переопределяет метод handle, который вызывается каждую единицу времени. При каждом

вызове метода handle известно текущее системное время и время генерации последнего транзакта. Когда разница между ними достигает NextTransact (время, через которое должен быть сгенерирован следующий транзакт), создается новый объект класса Transact и вызывается метод run (), после чего на основе изначально заданных параметров модели вновь вычисляется время NextTransact, через которое будет сгенерирован следующий транзакт.В системе может быть несколько генераторов (в данной задаче их два – для моделирования двух типов клиентов – мужчин и женщин). При этом каждый генератор также работает в отдельном потоке.

6.2. Реализация очереди обслуживания

В процессе моделирования перед каждым из мастеров образуется очередь из транзактов. Программно это достигается за счет реализации функции обслуживания в виде *synchronized*-блока, через который в заданный момент времени может проходить только один поток.

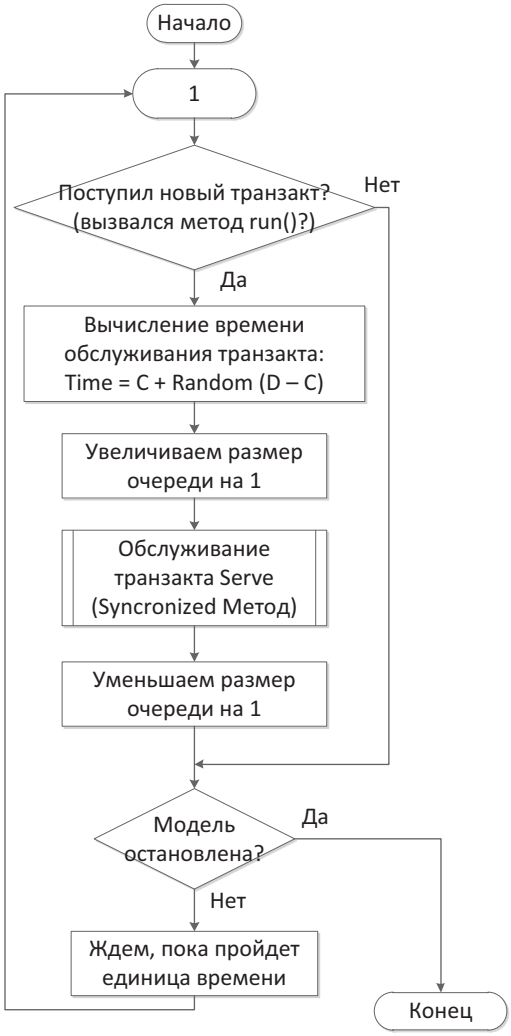

Рис. 5. Алгоритм реализации очереди

Для каждого вновь созданного транзакта вызывается метод run, после чего выполняется алгоритм, изображенный на рисунке 5. Алгоритм выполняется многими потоками параллельно, однако сам процесс обслуживания представлен в виде *synchronized*-метода *Serve,* который может выполняться лишь в один поток. Таким образом, перед этим блоком формируется очередь из транзактов. При этом не требуется создавать каких-либо структур данных, программно реализуя очередь, но всю работу удается возложить на среду выполнения и средства поддержки многопоточности языка Java. После остановки модели перестают поступать новые транзакты, а работа программы завершается, когда все клиенты, которые есть в системе, завершат обслуживание. Также реализуется постоянный контроль длины очереди. Во время подхода нового транзакта к очереди, соответствующая переменная увеличивается на 1, а после окончания обслуживания очередного клиента она уменьшается на 1. Чтобы избежать одновременного обращения двух потоков к общей области памяти, данная переменная реализована как атомарная (пакет *java.util.concurrent.atomic*).

6.3. Реализация сбора статистики

В процессе реализации сеанса моделирования программа осуществляет сбор статистики, а именно – среднее время обслуживания, загрузка каждого из мастеров, а также среднее время ожидания в очереди. Осуществляется это следующим образом: во время входа транзакта в очередь засекается момент времени; потом засекается момент времени, когда транзакт вышел из очереди. Зная время обслуживания данного транзакта, которое вычисляется динамически и данному транзакту оно известно, нетрудно вычислить время ожидания в очереди. Имея такую информацию, можно рассчитать среднее время обслуживания и среднее время ожидания в очереди. Поскольку известно количество клиентов, обслуженных каждым мастером, и среднее время обслуживания, простым перемножением можно получить время, в течение которого был занят мастер. Разделив его на полное модельное время, получим загрузку мастера в процентах, что выражено формулой (2):

$$\eta = \frac{N\mu}{T} \cdot 100\% \qquad (2),$$

где η - загрузка мастера в процентах, μ - среднее время обслуживания клиента, N –

количество клиентов, прошедших через мастера, T – моделируемый промежуток времени.

7. Классический подход к реализации задачи имитационного моделирования

Коротко рассмотрим логику работы интерпретатора GPSS, который использует другой подход для реализации алгоритма имитационного моделирования. В его основу положено продвижение транзактов от одного блока к другому. Когда движение данного транзакта блокируется, интерпретатор берется за продвижение другого транзакта. При этом интерпретатору необходимо определить, какой транзакт и куда двигать в каждый момент времени. Для этого интерпретатор GPSS рассматривает каждый транзакт как элемент одного или нескольких списков [5]. Одновременно транзакт является элементом некоторого модельного блока, где он находится в данный момент.

С каждым транзактом связано множество параметров, в том числе: номер транзакта; время движения; номер текущего блока; уровень приоритета; номер следующего блока. Эти параметры можно записать в виде вектора описания транзакта. Например, вектор (10, 510, 5, 0, 6) определяет транзакт с номером 10 и с нулевым уровнем приоритета, который находится в блоке с номером 5 и планирует войти в блок с номером 6 в 510-ю единицу модельного времени.

Работа интерпретатора GPSS с транзактами состоит в реализации трех фаз [5]:

1) фазы ввода;

2) фазы коррекции таймера модельного времени;

3) фазы просмотра списка текущих событий.

Для работы интерпретатора определяется таблица, где отражаются состояния списков текущих и будущих событий, соответствующие разным значениям модельного времени при реализации той или иной фазы моделирования. Пример такой таблицы представлен ниже:

Таблица 1

Состояния списков текущих и будущих событий

№	модельное время	список текущих событий	список будущих событий
1	0	Пусто	пусто
2	0	Пусто	[1,80,–,0,1], [2,1000,–,0,8]
3	80	[1,КМР,–,0,1]	[2,1000,–,0,8]
4	80	Пусто	[3,140,–,0,1], [1,150,5,0,6], [2,1000,–,0,8]
5	140	[3,КМР,–,0,1]	[1,150,5,0,6], [2,1000,–,0,8]
6	140	[3,КМР,2,0,3]	[1,150,5,0,6], [4,195,–,0,1], [2,1000,–,0,8]

7	150	[3,КМР,2,0,3], [1,КМР,5,0,6]	[4,195,–,0,1], [2,1000,–,0,8]
8	150	Пусто	[4,195,–,0,1], [3,195,5,0,6], [2,1000,–,0,8]
9	195	[4,КМР,–,0,1], [3,КМР,5,0,6]	[2,1000,–,0,8]
10	196	Пусто	[4,260,5,0,6], [5,270,–,0,1], [2,1000,–,0,8]
…	…	…	…
n-1	<1000	[…], […], …, […]	[2,1000,–,0,8], […], …, […]
n	1000	[…], […],…, […], [2,КМР,–,0,8]	
n+1	1000	[…], …, […]	[…],[…] ,…, [*k*,2000,–,0,8]

Подробнее логика работы интерпретатора GPSS представлена в работе [5].

8. Преимущества и особенности предложенного подхода

Проведем сравнение метода реализации имитационных моделей при помощи многопоточности и метода с использованием списка текущих и будущих событий.

К плюсам подхода с использованием многопоточного программирования можно отнести следующее:

1. Относительная простота реализации. При использовании списка текущих и будущих событий приходится отслеживать, чтобы все транзакты шли на выполнение в правильном порядке, в то время как при использовании многопоточности всю эту работу возможно возложить на среду выполнения.

2. Использование многопоточности позволяет задействовать все ядра процессора, т.е. достигается более полное использование ресурсов компьютера.

Однако у подхода с использованием многопоточности есть определенный недостаток: если среднее количество транзактов, одновременно присутствующих в модели, намного больше числа процессорных ядер в системе, то процессору придется многократно переключаться между потоками, что является затратной операцией, поскольку требует записи и восстановления контекста потока. Поэтому модель с большим числом одновременно присутствующих транзактов при реализации с использованием многопоточности будет очень требовательна к производительности процессора.

Анализируя преимущества и недостатки обоих реализаций, можно отметить, что для многоядерных систем, которые сейчас получают все большее распространение, наиболее эффективным будет подход с использованием многопоточности, но с использованием списка текущих и будущих событий. Суть такого синтеза заключается в том, чтобы равномерно разделить все транзакты по ядрам системы, используя многопоточность, а управлять всеми транзактами на конкретном ядре можно с использованием списка текущих и будущих событий. Реализовать такой подход

достаточно непросто, однако он позволит максимально использовать ресурсы многоядерных систем при построении сложных имитационных моделей.

9. Заключение

В процессе исследования возможности применения многопоточности для реализации задач имитационного моделирования было создано приложение, моделирующее процесс обслуживания клиентов в парикмахерской. Изучены средства языка программирования Java, необходимые для реализации предложенного подхода. Проведен анализ преимуществ и недостатков использования многопоточности при реализации задач имитационного моделирования по сравнению с традиционным подходом на основе списков текущих и будущих событий. По результатам анализа автором предложен подход, объединяющий преимущества обоих методов реализации задач имитационного моделирования.

Методы, предложенные в статье, могут оказаться полезными при реализации задач имитационного моделирования на многоядерных системах.

Список литературы

1. Lindberg P. Performance Obstacles for Threading: How do they affect OpenMP code? URL: http://software.intel.com/en-us/articles/performance-obstacles-for-threading-how-do-they-affect-openmp-code
2. Черненький В.М. Теоретические основы построения имитационного процесса. - М.: МГТУ им.Н.Э.Баумана, 2012.
3. Грегори Р. Эндрюс. Основы многопоточного, параллельного и распределенного программирования. Вильямс, 2003. – 512 с.
4. Brian Goetz and others. Java Concurrency in Practice. Addison-Wesley, USA. 2006. – 234 с.
5. Томашевский В., Жданова Е. Имитационное моделирование в среде GPSS. — М.: Бестселлер, 2003. — 416 с.