

Пенза 2026
Гурин Е.И.
Протокол заседания ГЭК от 10.06.2026 № 7

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

**ПРОГРАММА ОЦЕНКИ ВЕРОЯТНОСТНО-
ВРЕМЕННЫХ ХАРАКТЕРИСТИК
МНОГОПРОЦЕССОРНЫХ СИСТЕМ МЕТОДОМ
СТОХАСТИЧЕСКИХ СЕТЕЙ МАССОВОГО
ОБСЛУЖИВАНИЯ**

Студент

Петкилева Алёна Алексеевна
(подпись, дата) (ФИО полностью)

Руководитель

Бикташев Р.А.
(подпись, дата) (фамилия, инициалы)

Консультанты: по

(наименование раздела)(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

А.С.Бычков
(подпись, дата) (фамилия, инициалы)

Рецензент

наз. УИТ 05.06.2026 И.В.Савастьянов
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

10 июня 2026 № 10)

Секретарь ГЭК

Е.И. Гурин

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 11 » 02 20 20 г.

ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студентка Петкилева Алёна Алексеевна гр. 21ВОЭ1 факультета Вычислительной техники

2. Тема работы: «Программа оценки вероятностно-временных характеристик
многопроцессорных систем методом стохастических сетей массового обслуживания»

Тема утверждена приказом ПГУ № 122/о от "11" 02 2026 г.

3. Руководитель работы к.т.н., профессор Бикташев Р.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать аналитические модели, основанные на стохастических сетях массового обслуживания, в состав которых входят процессоры (1..N).

Разработку моделей производить на основании параметров многопроцессорной вычислительной системы, получаемых из исходных данных, которые задаются пользователем. Расчет характеристик (время задержки, время ответа и др.) производить с использованием программы расчета. Для этого необходимо разработать программу оценки вероятностно-временных характеристик многопроцессорных систем методом стохастических сетей массового обслуживания и вывод характеристик в виде отчета. Приложение должно поддерживать ОС Linux. Программа должна обладать графическим интерфейсом, позволяющим редактировать систему, как с помощью графических элементов, так и с помощью матрицы вероятностей передач. Система должна включать в себя функционал сохранения и импорта проекта, расчета основных характеристик сети, а также вывода результата в файл.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги существующих программ расчета стохастических систем, изучить актуальные способы сохранения данных о проекте, изучить способы проектирования и разработки приложения. Также необходимо изучить функционал библиотек GTK+, Cairo.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области и анализ существующих аналогов	50	11.02.2026 - 21.02.2026	
2.	Анализ требований и постановка задачи	10	22.02.2026 - 26.02.2026	
3.	Выбор технологий, применяемых при проектировании	50	27.03.2026 - 12.03.2026	
4.	Описание алгоритмов	40	13.03.2026 - 02.04.2026	
5.	Разработка приложения «Программа оценки вероятностно-временных характеристик многопроцессорных систем методом стохастических сетей массового обслуживания»	150	03.04.2026 - 14.05.2026	
6.	Оформление пояснительной записки	24	15.05.2026 - 21.05.2026	

Дата выдачи задания "11" 02 2026 г.

Руководитель дипломной работы  11.02.26
(подпись, дата)

Задание к исполнению принял студент 
(подпись, дата)

Работу к защите допустить
Декан факультета  Л.Р. Рязанский
02.06.2026 (подпись, дата)

... (дата) ...
...
... заседания ГЭК от 12.06.2026 № 7 ...
Гурия Е.И.

Реферат

Пояснительная записка содержит 149 листов, 41 рисунок, 3 таблицы, 3 источника и 27 приложений.

СТОХАСТИЧЕСКАЯ СЕТЬ МАССОВОГО ОБСЛУЖИВАНИЯ, МНОГОПРОЦЕССОРНАЯ СИСТЕМА, ВЕРОЯТНОСТНО-ВРЕМЕННЫЕ ХАРАКТЕРИСТИКИ, СИСТЕМА МАССОВОГО ОБСЛУЖИВАНИЯ, ЯЗЫК ПРОГРАММИРОВАНИЯ C, БИБЛИОТЕКА GTK+ 3, БИБЛИОТЕКА CAIRO, ФОРМАТ JSON, ALT LINUX

Основной целью является разработка приложения для расчёта вероятностно-временных характеристик многопроцессорных систем на основе стохастических сетей массового обслуживания. Программа обеспечивает визуальное построение графа сети, задание параметров узлов, расчёт характеристик, сохранение и загрузку проектов.

Объектом разработки является приложение, реализующее аналитические модели многоканальных СМО с ограниченной и неограниченной очередью. Программа обеспечивает добавление, удаление и перемещение узлов, создание связей с вероятностями переходов, редактирование параметров узлов, расчёт итерационным методом, визуализацию графа и результатов.

Разработка проводилась в ОС ALT Linux на языке C с использованием библиотек GTK+ 3 (графический интерфейс) и Cairo (визуализация графа). Для хранения проектов используется формат JSON.

В результате разработано приложение, которое обеспечивает визуальное построение и редактирование графа сети массового обслуживания, расчёт характеристик для многоканальных СМО, сохранение и загрузку проектов, экспорт результатов в текстовый файл.

ПУ 1.090501.12.001 ПЗ

					ПГУ 1.090501.12.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Программа оценки вероятностно-временных характеристик многопроцессорных систем методом стохастических сетей массового обслуживания Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Петкилева А.А.		05.06.26					
Провер.		Бикташев Р.А.		05.06.26			3	85	
						ФВТ, каф. ВТ гр. 21ВОЭ1			
Н. Контр.		Бычков А.С.		05.06.26					
Утверд.		Митрохин М.А.		05.06.26					

Содержание

Перечень сокращений и обозначений.....	7
Введение	8
1 Анализ темы исследования	10
1.1 Анализ предметной области	10
1.2 Анализ существующих программных решений и аналогов.....	11
1.3 Постановка задачи на разработку	12
2 Выбор инструментов разработки и элементной базы.....	17
2.1 Выбор программных средств разработки	17
2.2.1 Выбор языка программирования.....	17
2.2.2 Выбор библиотеки для графического интерфейса.....	18
2.2 Обоснование выбора формата данных	18
3 Архитектура приложения.....	20
3.1 Взаимодействие модулей	21
3.2 Диаграммы последовательности сценариев работы программы	23
3.2.1 Сценарий добавления нового узла СМО	23
3.2.2 Сценарий редактирования параметров узла.....	24
3.2.3 Сценарий создания связи между узлами	26
3.2.4 Сценарий выполнения расчёта и отображения результатов	28
3.2.5 Сценарий сохранения и загрузка проекта	31
4 Описание программного кода.....	34
4.1 Модель данных	34
4.2 Управление режимами редактирования	35
4.3 Обработка событий мыши и визуализация	36
4.4 Реализация расчёта характеристик СМО.....	39
4.5 Сохранение и загрузка проекта в формате JSON	42

4.6 Хранение и использование глобальных параметров	43
5 Тестирование	45
5.1 Проверка создания и редактирования графа сети	45
5.2 Проверка устойчивости при некорректных действиях пользователя	52
5.3 Тестирование расчёта характеристик	55
5.4 Оптимизация параметров системы.....	59
5.5 Выводы по тестированию.....	62
6 Инструкция по работе с программой	64
6.1 Установка и запуск программы	64
6.2 Построение модели	66
6.3 Расчёт характеристик.....	71
6.4 Сохранение и загрузка проекта	72
7 Экономика.....	73
8 Безопасность жизнедеятельности.....	77
8.1 Организация рабочего места разработчика.....	77
8.2 Меры пожарной безопасности при работе с компьютером.....	77
8.3 Факторы утомления и методы их снижения	78
8.4 Требования к освещению и воздушной среде.....	79
8.5 Рекомендации по организации рабочего времени.....	80
Заключение	81
Список использованных источников	85
Приложение А – UML-диаграммы.....	86
Приложение Б – Листинг программы	96
Приложение В – Презентация.....	136

Перечень сокращений и обозначений

ВВХ (вероятностно-временные характеристики) – показатели, описывающие случайные процессы в системах массового обслуживания (время ожидания, длина очереди, время пребывания заявки)

ОС (операционная система) – операционная система, комплекс программ, обеспечивающий управление аппаратными ресурсами компьютера и запуск прикладных программ

СеМО (сеть массового обслуживания) – совокупность взаимосвязанных систем массового обслуживания, по которым циркулируют заявки

СМО (система массового обслуживания) – объект, предназначенный для обработки поступающих заявок (запросов, требований) с возможным образованием очереди

ТМО (теория массового обслуживания) – раздел теории вероятностей, изучающий системы с потоками случайных событий и очередями

GNU (GNU's Not Unix) – проект по разработке свободного программного обеспечения, в рамках которого созданы многие инструменты для Linux.

GCC (GNU Compiler Collection) – набор компиляторов для различных языков программирования, включая C

GTK (GIMP Toolkit) – кроссплатформенная библиотека для создания графических пользовательских интерфейсов

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на синтаксисе JavaScript, используемый для хранения и передачи структурированных данных

М/М/м – многоканальная система массового обслуживания с неограниченной очередью, пуассоновским входящим потоком и экспоненциальным временем обслуживания

М/М/м/К – многоканальная система массового обслуживания с ограниченной очередью (максимальное число заявок в системе не превышает К)

					ПГУ 1.090501.12.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Во многих областях производства, бытового обслуживания, экономики и финансов важную роль играют системы специального вида, реализующие многократное выполнение однотипных задач. Подобные системы называют системами массового обслуживания (СМО). Они повсеместно встречаются в транспорте, промышленном производстве, образовании, медицине, военном деле, торговле, телефонии, компьютерных сетях и многих других сферах человеческой деятельности.

Системы и сети массового обслуживания являются объектами изучения теории массового обслуживания (ТМО) – одного из бурно развивающихся разделов теории вероятностей. В качестве систем массового обслуживания и состоящих из них сетей массового обслуживания (СеМО) рассматриваются разнообразные системы, предназначенные для обслуживания массового потока требований случайного характера. К настоящему времени разработаны относительно простые математические модели, позволяющие изучать многие внешне различающиеся реально протекающие процессы обслуживания. Конечная цель развиваемых в ТМО методов состоит в отыскании рациональных структуры и параметров обслуживающей системы, организации обслуживания, обеспечивающих заданное его качество [1].

Одной из важнейших современных областей применения теории массового обслуживания являются вычислительные системы, которые неуклонно движутся по пути наращивания вычислительной мощности за счёт увеличения количества процессорных элементов. Многопроцессорные системы стали основой как высокопроизводительных серверных решений, так и персональных компьютеров.

Актуальность темы обусловлена необходимостью разработки инструментария для исследования вероятностно-временных характеристик многопроцессорных систем с учётом конкурентного доступа к общей внешней

					ПГУ 1.090501.12.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

памяти и контроллерам ввода/вывода. Стохастические сети массового обслуживания являются наиболее подходящим математическим аппаратом для описания процессов, протекающих в таких системах, так как они позволяют моделировать параллелизм, очереди и случайный характер поступления и обработки задач. Создание программного приложения, реализующего расчёт таких сетей, позволит инженерам и исследователям оперативно получать оценки времени отклика, задержек и загрузки устройств, принимая обоснованные решения по оптимизации структуры вычислительной системы [2].

Целью данной работы является исследование вероятностно-временных характеристик многопроцессорных вычислительных систем путём разработки и программной реализации аналитических моделей на основе стохастических сетей массового обслуживания.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ многопроцессорных вычислительных систем как объектов массового обслуживания и параметры функционирования;
- разработать аналитические модели, описывающие функционирование вычислительной системы с переменным числом процессоров;
- создать приложение с графическим интерфейсом, которое позволяет пользователю задавать исходные параметры системы, выполнять расчёт её характеристик (времени задержки, времени отклика и др.) и визуализировать полученные результаты в виде отчётов;
- реализовать функционал редактирования структуры сети с помощью графических элементов и задания матрицы вероятностей передач, а также обеспечить сохранение и загрузку проектов.

Объектом исследования выступают многопроцессорные вычислительные системы, а предметом исследования – вероятностно-временные характеристики их функционирования, получаемые методом стохастических сетей массового обслуживания.

1 Анализ темы исследования

1.1 Анализ предметной области

Многопроцессорные вычислительные системы представляют собой сложный комплекс аппаратных средств, предназначенный для параллельной обработки данных. Ключевой особенностью таких систем является наличие нескольких процессоров, разделяющих общие ресурсы: оперативную (внешнюю) память и подсистему ввода/вывода, включающую соответствующие контроллеры. Важно отметить, что многопроцессорная вычислительная система работает под управлением единой операционной системы, общей для всех процессоров, что обеспечивает согласованность обработки данных и распределения нагрузки [3].

В многопроцессорной системе процессоры обращаются к памяти и устройствам ввода-вывода с запросами. Когда несколько процессоров одновременно пытаются получить доступ к одному и тому же устройству, возникают конфликты, и запросы выстраиваются в очередь. В теории массового обслуживания такие запросы называют заявками, а их выполнение - обслуживанием.

Метод стохастических сетей массового обслуживания позволяет адекватно описать данное поведение. В терминах СеМО каждый процессор, контроллер и блок памяти представляется в виде узла. Заявки перемещаются от узла к узлу в соответствии с некоторыми вероятностными правилами.

Для построения корректной модели необходимо определить следующие параметры:

- количество процессоров (N) определяет какое количество процессоров одновременно обрабатывают задачи;
- интенсивность поступления заявок – это частота генерации запросов каждым процессором;
- интенсивность обслуживания – это скорость обработки запросов

процессорами, памятью и контроллерами;

– матрица вероятностей передач описывает логику работы системы, т.е. вероятность того, что после обработки на одном устройстве заявка перейдет на другое или покинет систему.

При увеличении числа процессоров модель сети массового обслуживания сильно усложняется – растёт количество возможных состояний системы. Поэтому для расчёта характеристик (средней длины очередей, загрузки устройств и времени отклика) требуются эффективные вычислительные методы.

1.2 Анализ существующих программных решений и аналогов

На сегодняшний день существует ряд программных продуктов, позволяющих в той или иной степени моделировать вычислительные системы и сети массового обслуживания. Их анализ необходим для определения требований к разрабатываемому приложению и выявления его конкурентных преимуществ.

Среди данных программных средств хотелось бы в первую очередь выделить среду AnyLogic. Данное приложение является мощным инструментом как для традиционных обслуживающих систем, так и их разновидностей, требующих таких современных средств, как агентное моделирование или системная динамика. AnyLogic позволяет строить сложные модели, ориентированные на широкий класс задач, однако является коммерческим продуктом и требует значительных временных затрат на освоение и лицензирование. Кроме того, существует возможность самостоятельной доработки моделей и написания собственного кода средствами Java. В связи с этим, кроме аналитического аппарата, в литературе уделяется внимание особенностям получения вероятностно-временных характеристик с помощью средств имитационного моделирования AnyLogic. Тем не менее, отсутствие специализации на многопроцессорных архитектурах ограничивает его применение для решения узкоспециализированных задач данной области.

GPSS World – специализированный язык и среда для имитационного моделирования систем массового обслуживания. Обладает мощными вычислительными возможностями, но использует устаревший интерфейс и не предоставляет современных средств визуализации результатов в виде диаграмм и графов. Требуем от пользователя написания программы на языке GPSS (General Purpose System Simulation – система моделирования общего назначения), что создаёт дополнительный порог вхождения для исследователей, не обладающих глубокими навыками программирования.

MATLAB + Simulink – представляет собой мощную вычислительную среду, позволяющую реализовывать аналитические модели. Однако она не ориентирована на визуальное конструирование СМО и требует глубоких знаний программирования в среде MATLAB. Является коммерческим продуктом с высокой стоимостью лицензии, что делает его менее доступным для широкого круга пользователей.

JMT (Java Modelling Tools) – набор приложений для оценки производительности, моделирования систем, планирования параметров и исследования характеристик систем на основе моделей массового обслуживания. Предлагает графический интерфейс для построения сетей массового обслуживания и широкий набор алгоритмов расчёта. Однако его архитектура ориентирована на общие СМО и не предоставляет специализированных средств для моделирования многопроцессорных систем с учётом специфики работы контроллеров ввода/вывода.

1.3 Постановка задачи на разработку

Проведённый анализ предметной области и существующих программных аналогов показал, что, несмотря на наличие универсальных инструментов моделирования (AnyLogic, JMT), отсутствует специализированное решение, ориентированное на исследование многопроцессорных вычислительных систем с возможностью гибкой настройки их архитектуры (количество процессоров, наличие контроллеров ввода/вывода) и наглядной визуализацией результатов.

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

Существующие продукты либо требуют написания сложного программного кода, либо учитывают особенности доступа процессоров к общей памяти.

В связи с этим целью данной дипломной работы является разработка приложения, которая реализует аналитические модели для расчёта вероятностно-временных характеристик многопроцессорных систем на основе стохастических сетей массового обслуживания.

Приложение «Программа расчета характеристик стохастических сетей массового обслуживания» (далее – Приложение) предназначено для инженеров-исследователей, студентов и специалистов в области вычислительной техники. Оно позволяет на этапе проектирования или модернизации оценить производительность многопроцессорной системы, не прибегая к дорогостоящим испытаниям на реальном оборудовании.

Приложение должно предоставлять пользователю следующий функционал:

Задание структуры и параметров системы:

- добавление узлов (каждый узел может представлять процессор, контроллер или устройство памяти);
- задание для каждого узла параметров обслуживания (времени обработки заявки, количества каналов, типа очереди и её размера);
- создание связей между узлами с указанием вероятностей переходов.

Расчёт вероятностно-временных характеристик:

- вычисление среднего времени отклика системы;
- вычисление средней длины очереди к каждому устройству;
- вычисление коэффициента загрузки каждого устройства;
- вычисление среднего времени ожидания в очереди к каждому устройству.

Визуализация и экспорт

- структура моделируемой сети отображается в виде графа;

- вывод результатов расчёта в табличном виде;
- экспорт результатов в текстовый;
- сохранение и загрузка проекта (конфигурации сети) в формате JSON.

Требования к интерфейсу:

- интерфейс должен быть интуитивно понятным и соответствовать принципам человеко-машинного взаимодействия;
- графическое окно приложения должно обеспечивать панель ввода параметров, область вывода результатов, меню для работы с файлами проекта (сохранение, загрузка, экспорт).

В основу разрабатываемой программы заложены следующие исходные условия, соответствующие классическим сетям массового обслуживания:

- время обслуживания заявок распределено экспоненциально;
- дисциплина обслуживания во всех узлах - FIFO (First In, First Out).

Диаграмма вариантов использования, которая представлена на рисунке 1, иллюстрирует взаимодействие пользователя с системой и основные функциональные блоки приложения.

Таким образом, в результате выполнения дипломной работы должен быть создан законченный программный продукт, автоматизирующий процесс исследования многопроцессорных систем и позволяющий наглядно демонстрировать влияние архитектурных параметров (количество процессоров, скорость памяти) на итоговую производительность.

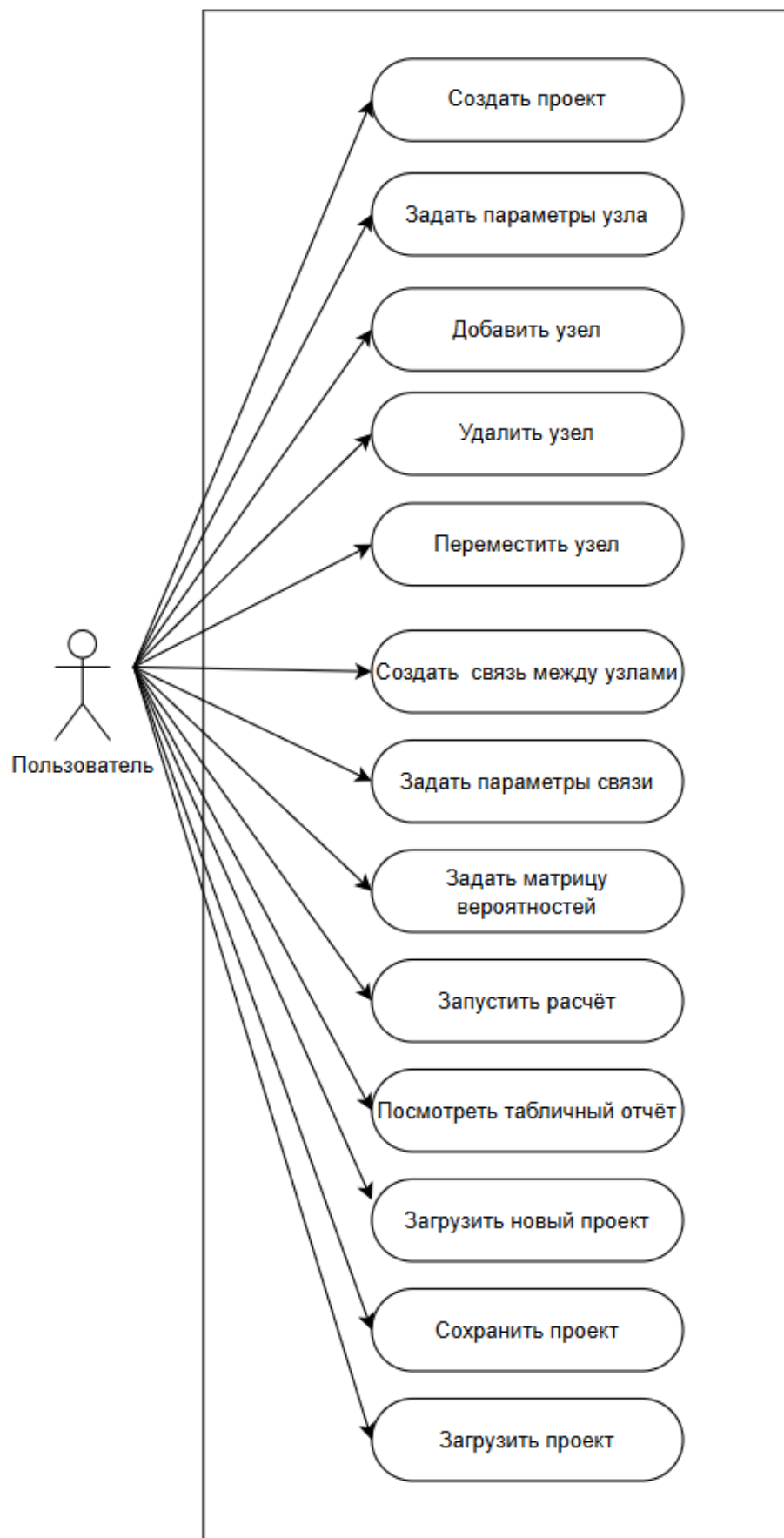


Рисунок 1 – Диаграмма вариантов использования

2 Выбор инструментов разработки и элементной базы

Чтобы успешно решить поставленные задачи, нужно выбрать программные средства разработки и аппаратные компоненты, чьи характеристики будут заложены в математические модели.

2.1 Выбор программных средств разработки

Для реализации программного продукта, выполняющего расчёт характеристик сетей массового обслуживания, необходим выбор языка программирования, среды разработки и дополнительных библиотек. Критериями выбора являются: производительность, кроссплатформенность (с фокусом на ОС Linux), наличие средств для построения графического интерфейса и удобство реализации математических алгоритмов.

2.2.1 Выбор языка программирования

Исходя из сформулированных требований, в качестве основного языка программирования выбирается C. Данный выбор обусловлен следующими факторами:

- C является компилируемым языком, обеспечивающим максимальную эффективность при реализации алгоритмов решения систем линейных уравнений большой размерности, возникающих при расчёте стохастических сетей;
- язык C позволяет тонко управлять использованием памяти и процессорного времени, что критически важно для вычислительно сложных задач;
- код на C может быть скомпилирован для любой платформы, включая все дистрибутивы Linux;
- стандартная библиотека C предоставляет необходимый минимум для работы с вводом/выводом и математическими вычислениями;
- C является родным языком для Linux/Unix-систем, что обеспечивает наилучшую интеграцию с операционной системой [4].

2.2.2 Выбор библиотеки для графического интерфейса

Для создания графического интерфейса пользователя и визуализации графов сети необходим выбор соответствующей библиотеки, ориентированной на работу в среде Linux.

В качестве базовой библиотеки выбирается GTK 3 – полноценный кроссплатформенный инструментарий, написанный на C и являющийся стандартом для многих Linux-приложений. Имеет средства для рисования (Cairo) и поддерживает создание интерфейсов любой сложности.

Выбор обусловлен следующими причинами:

- GTK написан на C и предоставляет API на C, что идеально сочетается с выбранным языком разработки;
- библиотека Cairo, входящая в экосистему GTK, позволяет отрисовывать графы сети, диаграммы и графики с высоким качеством.

Для отображения диаграмм и графиков будет использоваться Cairo как основной инструмент рисования.

В качестве среды разработки выбирается Visual Studio Code с набором расширений для языка C. Данный выбор обоснован следующими факторами:

- VS Code работает в Linux так же хорошо, как и в Windows;
- наличие плагинов для автодополнения, отладки и статического анализа кода;
- VS Code распространяется с открытой лицензией;
- возможность настройки сборки проекта через стандартные для Linux инструменты [5].

2.2 Обоснование выбора формата данных

Для хранения проектов моделирования выбирается формат JSON (JavaScript Object Notation). Данный выбор обусловлен тем, что:

- файл конфигурации можно просмотреть и отредактировать в любом текстовом редакторе без специализированного программного обеспечения;

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		17

– формат позволяет естественным образом описывать иерархические структуры: список узлов (процессоры, память), матрицу вероятностей (двумерный массив), глобальные параметры (тип сети, количество заявок).

– библиотеки для работы с JSON существуют для языка C (json-c, Jansson), что упрощает их интеграцию в проект [6].

Таким образом, выбранные программные и аппаратные компоненты в совокупности обеспечивают необходимый фундамент для разработки приложения на языке C под операционную систему Linux, способного решать поставленные задачи по исследованию многопроцессорных систем.

3 Архитектура приложения

Диаграмма последовательностей представлена на рисунке 2.

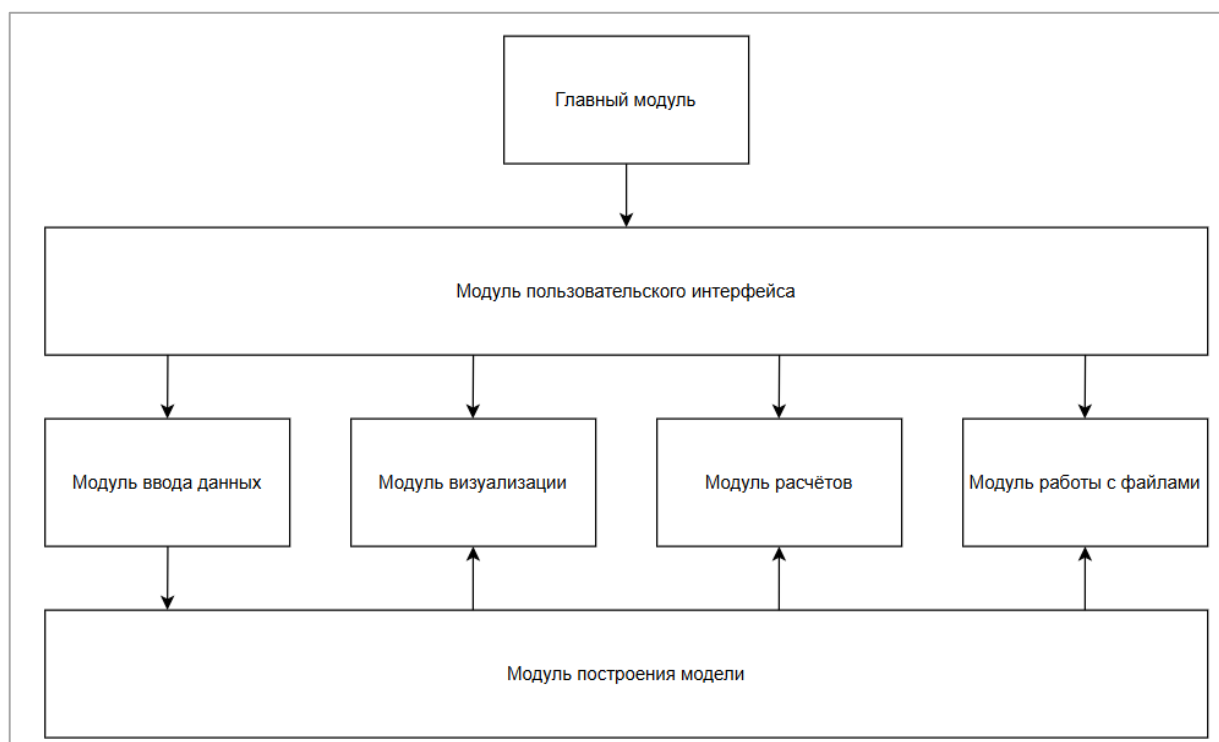


Рисунок 2 – Диаграмма последовательности

Разрабатываемое приложение представляет собой графическую программу для моделирования и расчёта характеристик стохастических сетей массового обслуживания. Данное приложение строится по модульному принципу. Каждый модуль решает свою задачу, что делает код понятнее и удобнее для доработки.

Архитектура приложения включает следующие модули: главный модуль, модуль пользовательского интерфейса, модуль ввода данных, модуль визуализации, модуль расчётов и модуль работы с файлами. Пользовательский интерфейс управляет работой остальных компонентов.

Главный модуль выполняет инициализацию GTK, создаёт главное окно приложения, размещает панель инструментов, меню и рабочую область, а также подключает обработчики событий.

Модуль пользовательского интерфейса отвечает за обработку нажатий кнопок и выбор пунктов меню. Он управляет режимами редактирования

(добавление и удаление узлов и связей, перемещение узлов), открывает диалоговые окна для ввода параметров и запускает расчёт. Через этот модуль пользователь взаимодействует со всеми функциями программы.

Модуль визуализации обеспечивает отображение моделируемой сети. Он рисует узлы в виде окружностей с подписями, связи между ними в виде линий со стрелками, а также отображает веса связей (вероятности переходов).

Модуль расчётов реализует математические алгоритмы. Он вычисляет интенсивности потоков заявок между узлами на основе заданной матрицы вероятностей, затем для каждого узла рассчитывает характеристики по формулам многоканальной СМО (коэффициент загрузки, среднюю длину очереди, среднее время ожидания и пребывания заявки).

Модуль ввода данных предоставляет диалоговые окна для настройки модели. Через него пользователь задаёт параметры узлов (наименование, время обслуживания, количество каналов, наличие и размер очереди), вводит вероятности переходов при создании связей, задаёт интенсивность входящего потока, а также редактирует матрицу вероятностей передач в табличном виде.

Модуль работы с файлами отвечает за сохранение и загрузку проектов в формате JSON. При сохранении записываются структура сети (координаты узлов), параметры всех узлов и связи с весами. При загрузке проект восстанавливается в том же состоянии. Кроме того, модуль сохраняет результаты расчёта в текстовый файл для последующего анализа.

3.1 Взаимодействие модулей

Пользователь работает с графическим интерфейсом через кнопки, которые располагаются в верхней части экрана.

Взаимодействие между модулями строится вокруг модуля построения модели, который является центральным хранилищем данных. Остальные модули либо записывают в него информацию, либо запрашивают её для выполнения своих задач.

При запуске программы главный модуль инициализирует GTK, создаёт

					ПГУ 1.090501.12.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

главное окно, размещает панель инструментов, меню и рабочую область. Затем он подключает обработчики событий к элементам интерфейса и передаёт управление модулю пользовательского интерфейса, который начинает ожидать действий пользователя.

Когда пользователь добавляет новый узел на рабочую область или изменяет параметры существующего, модуль пользовательского интерфейса вызывает модуль ввода данных, который открывает диалоговое окно.

После того как пользователь ввёл параметры (наименование узла, время обслуживания, количество каналов и т.д.), модуль ввода данных записывает эти значения в модуль построения модели. Модель хранит полную информацию о текущем состоянии сети: координаты узлов, их параметры, связи между ними.

После любого изменения модели (добавление или удаление узлов, перемещение, изменение связей) модуль пользовательского интерфейса даёт команду модулю визуализации перерисовать рабочую область. Получив команду, модуль визуализации запрашивает у модуля построения модели необходимые данные (координаты всех узлов, список связей между узлами, вероятности переходов). После того как модуль визуализации получил эти данные, он отрисовывает граф сети в рабочей области. Узлы отрисовываются в виде окружностей, связи в виде линий со стрелками, отображает веса связей.

Когда пользователь нажимает кнопку «Рассчитать», модуль пользовательского интерфейса запускает модуль расчётов. Модуль расчётов сначала запрашивает у модуля построения модели все необходимые данные, такие как количество узлов в сети, параметры каждого узла (время обслуживания, количество каналов, размер очереди), матрицу вероятностей переходов между узлами, интенсивность входящего потока.

Затем модуль расчётов выполняет вычисления. Определяет интенсивности потоков заявок между узлами, рассчитывает для каждого узла характеристики (коэффициент загрузки, среднюю длину очереди, среднее время ожидания и пребывания заявки). После завершения расчёта модуль расчётов записывает

полученные результаты обратно в модуль построения модели.

После расчёта пользователь может открыть отчёт. Для отчёта в виде текстового файла модуль пользовательского интерфейса вызывает модуль работы с файлами, который запрашивает у модуля построения модели результаты расчёта и сохраняет их в текстовый файл.

Когда пользователь сохраняет проект, модуль пользовательского интерфейса вызывает модуль работы с файлами. Модуль работы с файлами запрашивает у модуля построения модели полное состояние сети. Затем он преобразовывает эти данные в формат JSON и записывает в файл.

3.2 Диаграммы последовательности сценариев работы программы

Для наглядного представления динамики взаимодействия модулей были разработаны диаграммы последовательности для основных сценариев работы программы. Каждая диаграмма отражает обмен сообщениями между компонентами системы во времени. Актёром во всех сценариях выступает пользователь.

3.2.1 Сценарий добавления нового узла СМО

Данный сценарий описывает процесс создания нового узла системы массового обслуживания на рабочем поле. Диаграмма данного сценария представлена на рисунке 3.

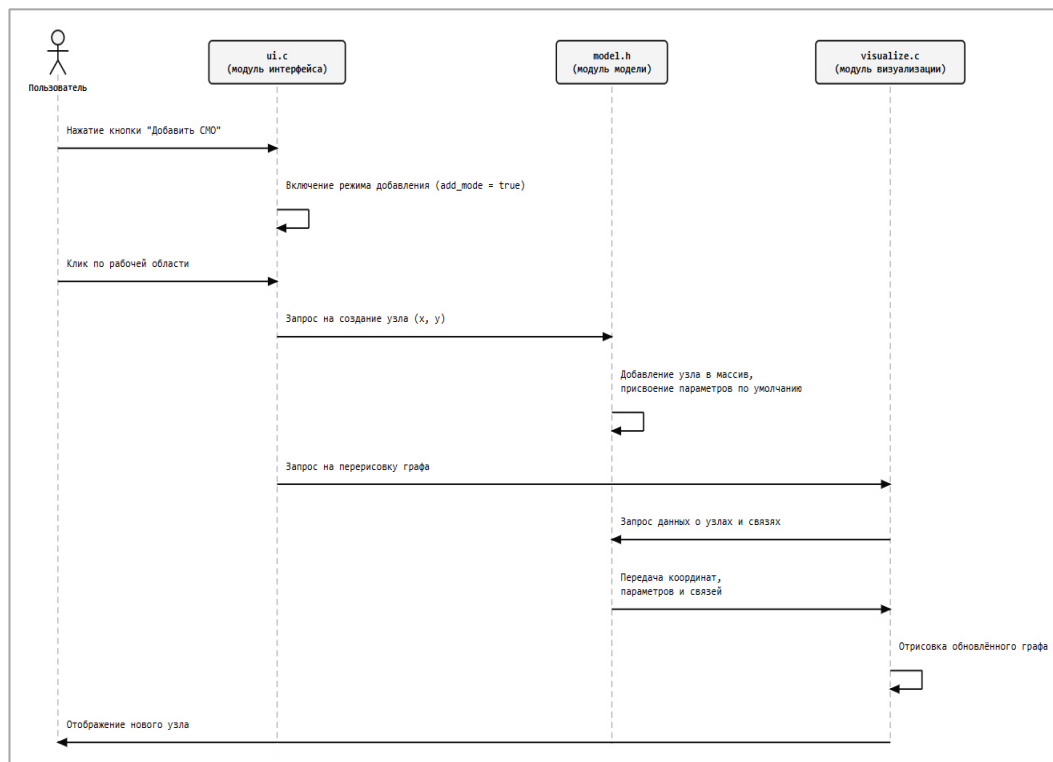


Рисунок 3 – Диаграмма создания нового узла

Пользователь инициирует добавление узла, наводя курсор на панель инструментов и нажимая кнопку «Добавить СМО». В этот момент модуль пользовательского интерфейса (ui.c) обрабатывает сигнал clicked от кнопки и вызывает функцию on_add_smo_clicked(), которая устанавливает флаг add_mode = true в структуре Circles (модуль модели). Это переводит программу в режим ожидания клика для размещения нового узла.

Затем пользователь перемещает курсор в область рисования (виджет GtkDrawingArea) и кликает левой кнопкой мыши в том месте, где должен появиться новый узел. Событие button-press-event перехватывается модулем визуализации (visualize.c), который вызывает функцию on_button_press(). Внутри этой функции программа проверяет текущий режим. Поскольку add_mode = true, выполняется ветка кода, отвечающая за добавление узла. Функция определяет координаты клика (event->x, event->y) и передаёт их в модуль модели (model.h).

Модуль модели добавляет новый узел в массив circles->x[] и circles->y[], увеличивает счётчик circles->count, а также инициализирует параметры узла

значениями по умолчанию: время обслуживания устанавливается равным 1,0, число каналов – 1, очередь по умолчанию не ограничена. После этого флаг `add_mode` сбрасывается в `false`.

После изменения модели модуль интерфейса (через тот же обработчик) вызывает функцию `gtk_widget_queue_draw()`, которая генерирует сигнал `draw` для области рисования. В ответ на этот сигнал модуль визуализации в функции `on_draw_event()` запрашивает у модуля модели актуальные списки узлов и связей, затем отрисовывает окружность, представляющую узел, с подписью ($S_0, S_1, \dots, S_n$) в указанных координатах. В результате пользователь видит новый узел в том месте, где был сделан клик.

3.2.2 Сценарий редактирования параметров узла

Сценарий иллюстрирует процесс изменения характеристик существующего узла. Диаграмма данного сценария представлена на рисунке 4.

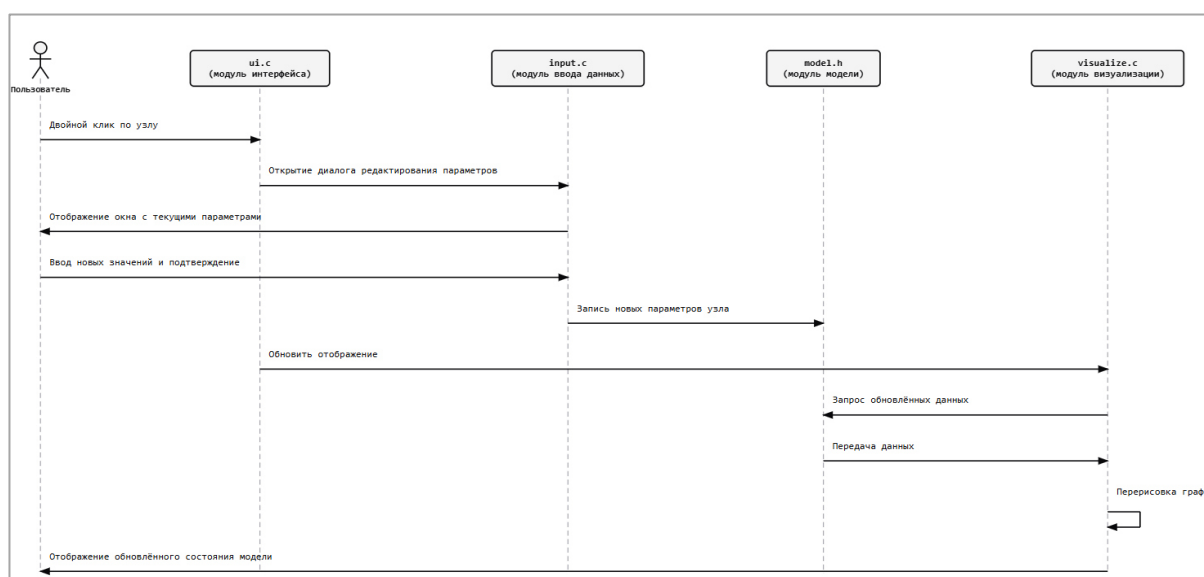


Рисунок 4 – Диаграмма редактирования параметров узла

Пользователь совершает двойной клик левой кнопкой мыши по узлу на графе сети. Событие `button-press-event` в модуле визуализации (`visualize.c`) обрабатывается функцией `on_button_press()`, которая сначала проверяет, что режимы добавления, удаления и перемещения узлов не активны, а затем определяет, что координаты клика попадают в область круга выбранного узла

(расстояние от точки клика до центра узла не превышает радиуса). Для этого функция перебирает массив `circles->x[]` и `circles->y[]`, вычисляя расстояние до каждого узла.

После того как индекс узла определён, программа проверяет его значение. Если узел имеет индекс 0 (то есть это корневой узел S0, который представляет источник входящего потока), то вызывается функция `prompt_user_for_intensity()` из модуля ввода данных (`input.c`), которая открывает диалоговое окно для изменения интенсивности входящего потока. Если же узел имеет индекс 1 или больше (обычные узлы СМО), то вызывается функция `show_circle_params_dialog()`, также из модуля ввода данных.

Модуль ввода данных создаёт и отображает диалоговое окно (`GtkDialog`) с полями ввода, заполненными текущими значениями из структуры `CircleParams` для выбранного узла. Пользователь видит текущие значения наименования узла, времени обслуживания, числа каналов, а также может установить или снять флаг ограниченной очереди и указать размер очереди. Пользователь изменяет нужные значения в полях ввода и нажимает кнопку «ОК» в диалоговом окне.

Модуль ввода данных считывает введённые значения из полей, преобразует их в соответствующие типы данных (с помощью функций `strtod` для чисел с плавающей точкой и `atoi` для целых чисел), выполняет контроль ввода (например, время обслуживания и число каналов должны быть больше нуля) и записывает новые значения обратно в структуру `CircleParams` выбранного узла в модуле модели (`model.h`). После этого устанавливается флаг `project_modified = true`, сигнализирующий о том, что в проект были внесены изменения.

После закрытия диалогового окна модуль визуализации получает команду на перерисовку рабочей области. Пользователь видит, что диалоговое окно закрылось, а граф сети остаётся на месте, но теперь узлы содержат новые параметры, которые будут использоваться при последующем расчёте характеристик.

3.2.3 Сценарий создания связи между узлами

Сценарий описывает процесс установления маршрута перехода заявок между двумя узлами. Диаграмма данного сценария представлена на рисунке 5.

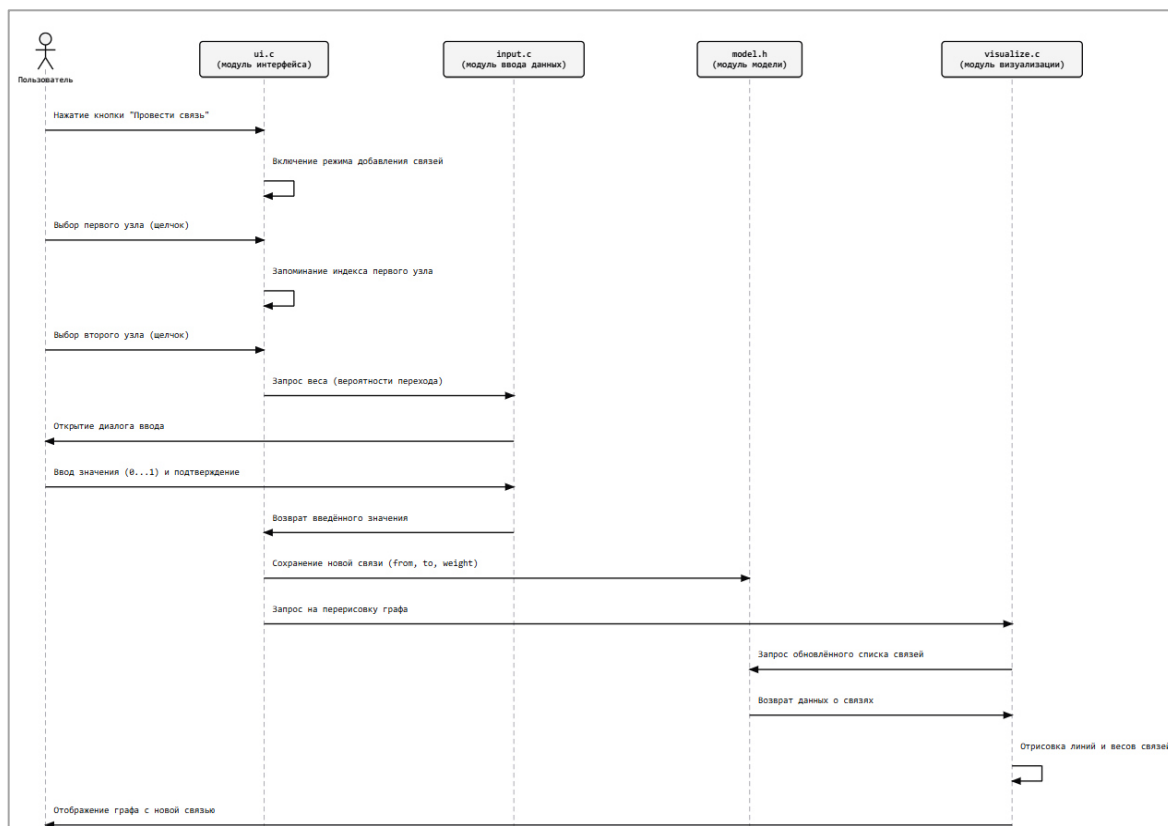


Рисунок 5 – Диаграмма создания связи между узлами

Пользователь начинает с нажатия кнопки «Провести связь» на панели инструментов. Модуль интерфейса (ui.c) обрабатывает это событие и вызывает функцию `enable_add_line_mode()`, которая устанавливает флаг `add_line_mode = true` в структуре `Circles`, а также инициализирует переменную `line_first_circle` значением -1. Теперь программа находится в режиме ожидания выбора двух узлов для создания связи.

Затем пользователь кликает по первому узлу (узлу-источнику). Модуль визуализации в функции `on_button_press()` обнаруживает, что флаг `add_line_mode` установлен в `true`, определяет индекс узла, по которому был произведён клик, и сохраняет этот индекс в переменную `line_first_circle`.

После этого пользователь кликает по второму узлу (узлу-приёмнику). Модуль визуализации снова перехватывает событие, видит, что `line_first_circle` уже содержит допустимый индекс, и определяет индекс второго узла. Поскольку оба узла выбраны, программа переходит к этапу запроса веса связи.

Для этого модуль визуализации вызывает функцию `prompt_user_for_weight()` из модуля ввода данных (`input.c`). Модуль ввода данных создаёт и отображает диалоговое окно с полем для ввода вещественного числа. Пользователь вводит вероятность перехода - значение от 0 до 1, которое определяет, какая доля заявок из первого узла переходит во второй. Введённое значение может быть, например, 0,7 (что означает 70% заявок). После ввода пользователь нажимает кнопку «ОК».

Модуль ввода данных возвращает введённое значение в модуль визуализации, который затем вызывает функцию добавления связи в модуле модели (`model.h`). Модуль модели проверяет, не превышен ли максимально допустимый лимит связей (`MAX_LINES`), и если нет, добавляет новую структуру `Line` в массив `circles->lines[]`, заполняя поля `from_index` (индекс первого узла), `to_index` (индекс второго узла) и `weight` (введённая вероятность перехода). Счётчик `line_count` увеличивается на единицу.

После успешного добавления связи модуль модели возвращает управление модулю визуализации, который устанавливает флаг `add_line_mode = false` и вызывает перерисовку рабочей области. В функции `on_draw_event()` модуль визуализации запрашивает у модуля модели обновлённый список связей и для каждой из них отрисовывает линию от края одного узла до края другого (чтобы линия не заходила внутрь круга), стрелку на конце линии, указывающую направление перехода, и текстовую метку с вероятностью (например, «0,7») рядом с линией. В результате пользователь видит на графе новую связь, полностью готовую к использованию в расчётах.

3.2.4 Сценарий выполнения расчёта и отображения результатов

Данный сценарий описывает ключевой вычислительный процесс, в ходе которого программа на основе заданной пользователем конфигурации сети рассчитывает её основные характеристики и отображает результаты в наглядной форме. Диаграмма данного сценария представлена на рисунке 6.

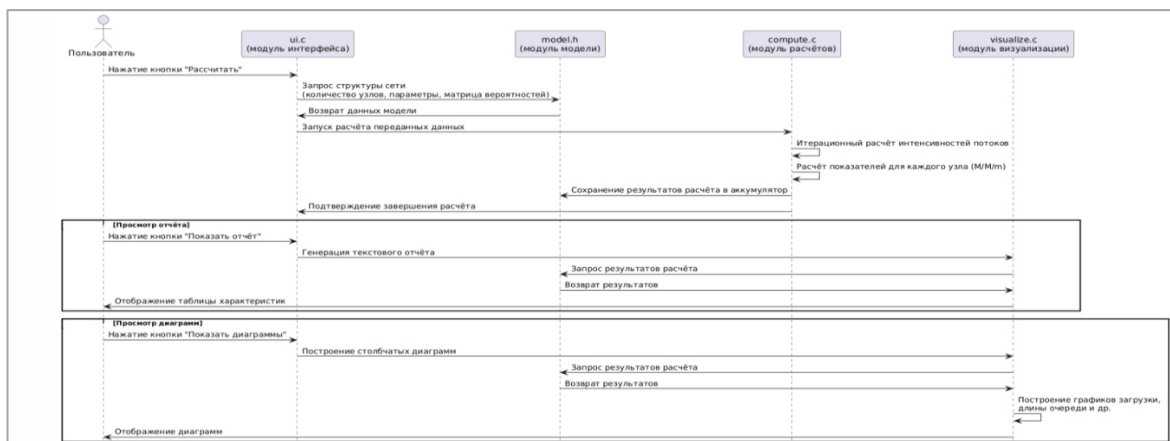


Рисунок 6 – Диаграмма выполнения расчёта и отображения результатов

Пользователь инициирует расчёт, нажимая кнопку «Рассчитать» на панели инструментов или выбирая соответствующий пункт в меню «Сеть». Модуль интерфейса (ui.c) обрабатывает это событие и вызывает функцию `on_calc_net_clicked()`. В этой функции программа сначала запрашивает у модуля модели (model.h) полную информацию о текущей конфигурации сети: количество узлов (count), параметры каждого узла (время обслуживания `service_time`, количество каналов `channel_count`, наличие и размер ограниченной очереди), а также список связей с весами (`lines[]` и `weight`), которые в совокупности образуют матрицу вероятностей переходов между узлами.

После получения всех необходимых данных модуль интерфейса вызывает функцию `calculate_network()` из модуля расчётов (compute.c), передавая ей указатель на структуру `Circles`, текущее значение интенсивности входящего потока (`input_intensity`) и индекс цикла (при обычном расчёте он равен 0). Управление переходит в вычислительный модуль.

Внутри `calculate_network()` программа сначала проверяет корректность матрицы вероятностей: для каждого узла вычисляется сумма весов всех исходящих связей, и если эта сумма значительно отличается от 1,0 (с некоторым допустимым отклонением), то на экран выводится предупреждение. Затем начинается расчёт интенсивностей потоков. Создаётся массив `lambda[MAX_CIRCLES]`, в котором `lambda[0]` устанавливается равным заданной интенсивности входящего потока. На каждой итерации для всех связей вычисляются новые значения интенсивностей по формуле `new_lambda[to] += lambda[from] * weight`, где `weight` - вероятность перехода. Этот процесс повторяется до тех пор, пока изменения значений интенсивностей на итерации не станут меньше заданной точности `epsilon = 1e-6`, либо до достижения максимального числа итераций (100). После достижения сходимости программа получает установившиеся значения интенсивностей потоков для каждого узла сети.

Далее для каждого узла программа вызывает функцию `calculate_mm_m()`, которая реализует математические формулы для многоканальной системы массового обслуживания. Если для узла установлен флаг `limited_queue = true`, то вычисления ведутся по формулам для системы с ограниченной очередью (М/М/м/К), в противном случае - для системы с неограниченной очередью (М/М/м). В ходе расчёта вычисляются следующие характеристики: коэффициент загрузки узла (`load_factor`) как отношение интенсивности входящего потока к максимальной пропускной способности; среднее число занятых каналов (`busy_channels`); средняя длина очереди (`avg_in_queue`); среднее число заявок в системе (`avg_in_system`, включая обслуживаемые и ожидающие); среднее время ожидания в очереди (`avg_wait_queue`) и среднее время пребывания заявки в системе (`avg_time_system`). Если коэффициент загрузки достигает или превышает 1,0 и при этом очередь не ограничена, система считается нестабильной, и пользователь получает соответствующее предупреждение.

После того как все характеристики рассчитаны, программа сохраняет полученные значения в структуру-аккумулятор results_acc в модуле модели, где они хранятся для последующего отображения. Модуль расчётов возвращает управление модулю интерфейса, который уведомляет пользователя о завершении расчёта (сообщение выводится в терминал).

Для просмотра результатов пользователь может нажать кнопку «Показать отчёт» на панели инструментов. В этом случае модуль интерфейса вызывает функцию save_results_to_file() из модуля работы с файлами (io.c). Эта функция запрашивает у модуля модели результаты расчёта из структуры results_acc, форматирует их в виде текстовой таблицы, включающей характеристики каждого узла (коэффициент передачи, интенсивность входящего потока, коэффициент загрузки, среднее число занятых каналов, среднее число заявок в системе, среднюю длину очереди, среднее время ожидания в очереди и среднее время пребывания в системе), и сохраняет в файл result.txt. Затем файл открывается в стандартном в стандартном приложении для просмотра текстовых файлов текстовых файлов, и пользователь видит подробный отчёт.

Также пользователь может просмотреть диаграммы характеристик, нажав кнопку «Показать диаграмму характеристик сети». В этом случае модуль интерфейса вызывает функцию show_diag_net_dialog() из модуля визуализации, которая создаёт новое диалоговое окно, запрашивает у модуля модели результаты расчёта и строит столбчатые диаграммы, отображающие загрузку каждого узла и длину очередей. Пользователь может выбирать, какие именно характеристики отображать, с помощью флажков в диалоговом окне.

3.2.5 Сценарий сохранения и загрузка проекта

Данный сценарий описывает работу модуля файлового ввода-вывода, который обеспечивает сохранение текущего состояния модели на диск и последующее восстановление из сохранённого файла. Диаграмма данного сценария представлена на рисунке 7.

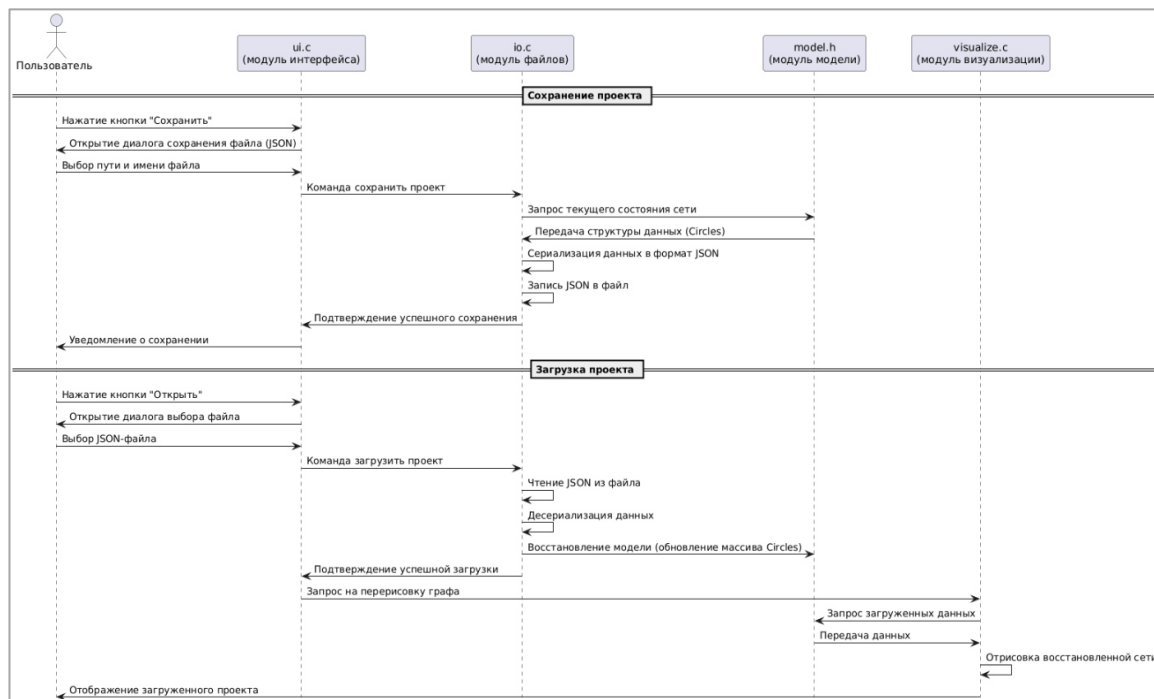


Рисунок 7 – Диаграмма сохранения и загрузка проекта

Когда пользователь желает сохранить текущий проект, он нажимает кнопку «Сохранить» на панели инструментов или выбирает пункт меню «Файл» → «Сохранить». Модуль интерфейса (ui.c) обрабатывает это событие и вызывает функцию `on_save_project_clicked()`. В этой функции программа создаёт диалоговое окно выбора файла (`GtkFileChooserDialog`), которое позволяет пользователю выбрать папку для сохранения и ввести имя файла. В диалоге предустановлен фильтр, отображающий только файлы с расширением `.json`, а в поле имени файла по умолчанию подставляется значение `project.json`. Пользователь выбирает нужное место на диске, при необходимости изменяет имя файла и подтверждает сохранение.

После подтверждения модуль интерфейса вызывает функцию `save_circles_to_file()` из модуля работы с файлами (io.c), передавая ей указатель на структуру `Circles` (модель) и выбранное имя файла. Модуль работы с файлами запрашивает у модуля модели (model.h) все данные, необходимые для сохранения. Затем начинается процесс записи данных в формате JSON с использованием библиотеки `json-glib`. Создаётся новый JSON-объект через

JsonBuilder, в который последовательно добавляются: версия формата данных, количество узлов, количество связей, массив координат узлов (каждый узел сохраняется как объект с полями x и y), массив связей (каждая связь сохраняется как объект с полями from, to и weight) и массив параметров узлов (каждый узел сохраняется как объект с полями name, service_time, channel_count, limited_queue, queue_size). После того как JSON-объект полностью сформирован, он записывается в файл на диск с помощью функции json_generator_to_file(). При успешном сохранении флаг project_modified сбрасывается в false, и пользователь получает уведомление о том, что проект сохранён.

Когда пользователь желает загрузить ранее сохранённый проект, он нажимает кнопку «Открыть» на панели инструментов или выбирает пункт меню «Файл» → «Открыть». Модуль интерфейса вызывает функцию on_open_project_clicked(), которая создаёт диалоговое окно выбора файла (аналогичное окну сохранения, но в режиме открытия). Пользователь находит нужный JSON-файл в файловой системе, выбирает его и подтверждает загрузку.

Модуль интерфейса вызывает функцию load_circles_from_file() из модуля работы с файлами (io.c), передавая ей имя выбранного файла и указатель на структуру Circles. Модуль работы с файлами открывает файл, считывает его содержимое и выполняет парсинг JSON с помощью JsonParser. Если парсинг прошёл успешно, программа сначала очищает текущую модель, вызывая clear_circles(), а затем извлекает сохранённые значения из JSON-объекта: восстанавливает количество узлов, их координаты, список связей с весами, а также параметры каждого узла (наименование, время обслуживания, число каналов, флаг ограниченной очереди и размер очереди). Все восстановленные данные записываются обратно в структуру Circles в модуле модели. В случае ошибки чтения или некорректного формата файла программа выводит диалоговое окно с сообщением об ошибке.

После успешной загрузки модуль работы с файлами возвращает управление модулю интерфейса, который затем вызывает перерисовку рабочей

области. Модуль визуализации запрашивает у модуля модели восстановленные данные и отрисовывает граф сети в том виде, в котором он был сохранён: узлы располагаются на тех же координатах, связи отображаются с теми же весами, параметры узлов соответствуют сохранённым значениям. В результате пользователь видит на экране полностью восстановленный проект и может продолжить работу с ним с того места, где остановился.

					ПГУ 1.090501.12.001 ПЗ	Лист
						33
Изм.	Лист	№ докум.	Подпись	Дата		

4 Описание программного кода

В данном разделе рассматриваются ключевые аспекты программной реализации описанной выше архитектуры. Приводится объяснение наиболее важных фрагментов кода, реализующих основные функции программы: управление моделью данных, обработку событий графического интерфейса, визуализацию графа сети, расчёт характеристик и работу с файлами.

4.1 Модель данных

В основе программы лежат структуры данных, описывающие элементы сети массового обслуживания. Все структуры объединены в центральной структуре Circles, которая хранит полную информацию о текущем состоянии модели.

```
#define MAX_CIRCLES 100
#define MAX_LINES 100
typedef struct {
    int from_index;        // индекс узла-источника
    int to_index;          // индекс узла-приёмника
    float weight;          // вероятность перехода
} Line;
typedef struct {
    char* circle_name;     // наименование узла
    double service_time;   // среднее время обслуживания
    int channel_count;     // количество каналов
    bool limited_queue;    // флаг ограниченной очереди
    int queue_size;        // максимальный размер очереди
} CircleParams;
typedef struct {
    double x[MAX_CIRCLES]; // координаты узлов по оси X
    double y[MAX_CIRCLES]; // координаты узлов по оси Y
    int count;             // количество узлов
    bool add_mode;         // режим добавления узлов
    bool del_mode;         // режим удаления узлов
```

```

bool move_mode;           // режим перемещения узлов
bool add_line_mode;       // режим добавления связей
Line lines[MAX_LINES];    // массив связей
int line_count;           // количество связей
CircleParams circle_params[MAX_CIRCLES]; //параметры узлов
} Circles;

```

Структура Line описывает, как заявки переходят из одного узла в другой и содержит индексы узла-источника (from_index), узла-приёмника (to_index) и вероятность перехода заявки (weight). Структура CircleParams хранит параметры отдельного узла СМО: наименование, время обслуживания (обратная величина к интенсивности обслуживания μ), количество каналов, флаг ограниченной очереди и размер очереди. Структура Circles объединяет все элементы модели: координаты узлов (для визуализации), их количество, флаги режимов редактирования, массив связей и массив параметров узлов.

4.2 Управление режимами редактирования

Взаимодействие пользователя с программой осуществляется через кнопки панели инструментов, которые переключают режимы редактирования. Каждая кнопка связана с функцией-обработчиком, изменяющей соответствующие флаги в структуре Circles.

```

void on_add_cmo_clicked(GtkButton* button, gpointer user_data)
{
    Circles* circles = (Circles*)user_data;
    circles->add_mode = true;
    circles->del_mode = false;
    circles->move_mode = false;
    circles->add_line_mode = false;
    circles->delLine_mode = false;
}

void on_del_cmo_clicked(GtkButton* button, gpointer user_data)
{
    Circles* circles = (Circles*)user_data;

```

```

    circles->del_mode = true;
    circles->add_mode = false;
    circles->move_mode = false;
    circles->add_line_mode = false;
    circles->delLine_mode = false;
}

```

При нажатии на кнопку «Добавить СМО» функция `on_add_smo_clicked()` устанавливает флаг `add_mode = true`, одновременно сбрасывая все остальные режимы. Аналогичным образом работают функции для удаления (`on_del_smo_clicked`), перемещения узлов (`on_move_smo_clicked`), добавления связей (`on_add_line_clicked`) и удаления связей (`on_del_line_clicked`). Такой подход гарантирует, что в каждый момент времени активен только один режим редактирования.

4.3 Обработка событий мыши и визуализация

Основная логика взаимодействия с рабочей областью реализована в модуле `visualize.c`. Функция `on_button_press()` обрабатывает нажатия мыши, определяет текущий режим и выполняет соответствующее действие.

```

gboolean on_button_press(GtkWidget* widget, GdkEventButton*
event, gpointer user_data) {
    Circles* circles = (Circles*)user_data;
    const double radius = 30.0;
    if (event->button == GDK_BUTTON_PRIMARY) {
        // Режим добавления узла
        if (circles->add_mode) {
            if (circles->count < MAX_CIRCLES) {
                circles->x[circles->count] = event->x;
                circles->y[circles->count] = event->y;
                circles->circle_params[circles-
>count].service_time = 1.0;
                circles->circle_params[circles-
>count].channel_count = 1;
                circles->count++;
            }
        }
    }
}

```

```

        circles->add_mode = false;
        gtk_widget_queue_draw(widget);
    }
    return TRUE;
}

// Режим удаления узла
else if (circles->del_mode) {
    for (int i = 0; i < circles->count; i++) {
        double dx = event->x - circles->x[i];
        double dy = event->y - circles->y[i];
        if (sqrt(dx*dx + dy*dy) <= radius) {
            delete_circle_and_lines(circles, i);
            circles->del_mode = false;
            gtk_widget_queue_draw(widget);
            break;
        }
    }
    return TRUE;
}

// Режим перемещения узла
else if (circles->move_mode) {
    for (int i = 0; i < circles->count; i++) {
        double dx = event->x - circles->x[i];
        double dy = event->y - circles->y[i];
        if (sqrt(dx*dx + dy*dy) <= radius) {
            circles->moving_index = i;
            return TRUE;
        }
    }
}

return FALSE;
}

```

При нажатии левой кнопки мыши функция проверяет текущий режим через соответствующие флаги. В режиме добавления (add_mode = true)

					ПГУ 1.090501.12.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

программа создаёт новый узел в координатах клика, инициализирует его параметры значениями по умолчанию (время обслуживания 1,0, один канал) и вызывает перерисовку области. В режиме удаления программа перебирает все существующие узлы, вычисляет расстояние от точки клика до центра каждого узла и при попадании в область круга (радиус 30 пикселей) вызывает функцию удаления узла и всех связанных с ним связей. В режиме перемещения программа запоминает индекс узла, по которому был произведён клик, для последующего перетаскивания.

```
gboolean on_draw_event(GtkWidget* widget, cairo_t* cr,
gpointer user_data) {
    Circles* circles = (Circles*)user_data;
    const double radius = 25.0;
    cairo_set_source_rgb(cr, 1, 1, 1);
    cairo_paint(cr);
    // Рисование связей
    for (int i = 0; i < circles->line_count; i++) {
        int from = circles->lines[i].from_index;
        int to = circles->lines[i].to_index;
        float weight = circles->lines[i].weight;
        double x1 = circles->x[from], y1 = circles->y[from];
        double x2 = circles->x[to], y2 = circles->y[to];
        // Вычисление направления и точек начала/конца линий
        double dx = x2 - x1, dy = y2 - y1;
        double dist = sqrt(dx*dx + dy*dy);
        double nx = dx / dist, ny = dy / dist;
        double start_x = x1 + nx * radius;
        double start_y = y1 + ny * radius;
        double end_x = x2 - nx * radius;
        double end_y = y2 - ny * radius;
        cairo_move_to(cr, start_x, start_y);
        cairo_line_to(cr, end_x, end_y);
        cairo_stroke(cr);
        // Рисование стрелки
```

```

        double angle = atan2(dy, dx);
        cairo_move_to(cr, end_x, end_y);
        cairo_line_to(cr, end_x - 10 * cos(angle - M_PI/6),
                      end_y - 10 * sin(angle - M_PI/6));
        cairo_line_to(cr, end_x - 10 * cos(angle + M_PI/6),
                      end_y - 10 * sin(angle + M_PI/6));
        cairo_close_path(cr);
        cairo_fill(cr);
    }
    // Рисование узлов
    for (int i = 0; i < circles->count; i++) {
        cairo_arc(cr, circles->x[i], circles->y[i], radius, 0,
2 * M_PI);

        cairo_fill_preserve(cr);
        cairo_stroke(cr);
        char label[16];
        snprintf(label, sizeof(label), "S%d", i);
        cairo_move_to(cr, circles->x[i] - 10, circles->y[i] +
5);

        cairo_show_text(cr, label);
    }
    return FALSE;
}

```

Функция `on_draw_event()` вызывается каждый раз при необходимости перерисовки рабочей области. Сначала она заливает фон белым цветом. Затем для каждой связи вычисляются координаты начала и конца линии (с учётом радиуса узлов, чтобы линия не заходила внутрь круга), после чего линия отрисовывается с помощью библиотеки Cairo. На конце каждой линии рисуется стрелка, направление которой вычисляется через арктангенс разности координат узлов. Пользователь видит связи как направленные линии с указанием вероятностей переходов. После отрисовки связей функция рисует узлы в виде окружностей с подписями S0, S1, S2 и т.д.

4.4 Реализация расчёта характеристик СМО

Модуль расчётов compute.c реализует аналитические алгоритмы для определения характеристик систем массового обслуживания. Основная вычислительная функция calculate_mm_m() реализует формулы Эрланга для многоканальной СМО.

```
CircleCalcResult calculate_mm_m(double lambda, double mu, int
m, bool limited_queue, int queue_size) {
    CircleCalcResult res = {0};
    double rho = lambda / (m * mu); // коэффициент загрузки
    if (limited_queue) {
        // М/М/м/К с ограниченной очередью
        int K = m + queue_size;
        // Вычисление суммы для P0
        double sum1 = 0.0;
        for (int n = 0; n < m; n++) {
            sum1 += pow(m * rho, n) / factorial(n);
        }
        double sum2 = (pow(m * rho, m) / factorial(m)) *
            (1 - pow(rho, K - m + 1)) / (1 - rho);
        double P0 = 1.0 / (sum1 + sum2);
        // Вычисление средней длины очереди
        double Lq = P0 * pow(m * rho, m) * rho /
            (factorial(m) * pow(1 - rho, 2));
        res.avg_in_queue = Lq;
        res.avg_wait_queue = Lq / lambda;
        res.avg_time_system = res.avg_wait_queue + 1.0 / mu;
        res.load_factor = rho;
    } else {
        // М/М/м с неограниченной очередью
        // Аналогичный расчёт...
    }
    return res;
}
```

Функция `calculate_mm_m()` принимает интенсивность входящего потока `lambda`, интенсивность обслуживания `mu` (обратная величина к времени обслуживания), количество каналов `m`, а также флаг ограниченной очереди и её размер. На основе этих параметров вычисляется коэффициент загрузки системы $\rho = \lambda / (m * \mu)$. Для системы с ограниченной очередью (М/М/м/К) расчёт ведётся по формулам Эрланга с учётом конечного числа мест в очереди. Вычисляются стационарные вероятности состояний P_0 , затем средняя длина очереди L_q и среднее время ожидания $W_q = L_q / \lambda$. Для системы с неограниченной очередью используется классическая формула Эрланга для многоканальной СМО.

```
void calculate_network(Circles* circles, double
current_intensity, int cycle_index) {
    double lambda[MAX_CIRCLES] = {0};
    lambda[0] = current_intensity;
    // Итерационный расчёт интенсивностей потоков
    for (int iter = 0; iter < 100; iter++) {
        double new_lambda[MAX_CIRCLES] = {0};
        new_lambda[0] = current_intensity;
        for (int i = 0; i < circles->line_count; i++) {
            int from = circles->lines[i].from_index;
            int to = circles->lines[i].to_index;
            double weight = circles->lines[i].weight;
            new_lambda[to] += lambda[from] * weight;
        }
        double max_diff = 0.0;
        for (int i = 1; i < circles->count; i++) {
            double diff = fabs(new_lambda[i] - lambda[i]);
            if (diff > max_diff) max_diff = diff;
            lambda[i] = new_lambda[i];
        }
        if (max_diff < 1e-6) break; // достигнута сходимость
    }
    // Расчёт характеристик для каждого узла
    for (int i = 0; i < circles->count; i++) {
```

```

double mu = 1.0 / circles->circle_params[i].service_time;

int m = circles->circle_params[i].channel_count;
CircleCalcResult res = calculate_mm_m(lambda[i], mu,
m, ...);

results_acc.results[i][cycle_index] = res;
}
}

```

Функция `calculate_network()` выполняет расчёт всей сети. Сначала итерационным методом вычисляются интенсивности потоков в каждом узле. Начальное значение `lambda[0]` задаётся пользователем (интенсивность входящего потока). На каждой итерации для каждой связи значение `new_lambda[to]` увеличивается на `lambda[from] * weight`. Процесс повторяется до тех пор, пока изменения интенсивностей не станут меньше заданной точности ($1 \cdot 10^{-6}$). После достижения сходимости для каждого узла вызывается функция `calculate_mm_m()`, и результаты сохраняются в аккумулятор `results_acc`.

4.5 Сохранение и загрузка проекта в формате JSON

Для сохранения и загрузки проектов используется формат JSON. Модуль `io.c` реализует функции сериализации и десериализации данных модели с использованием библиотеки `json-glib`.

```

bool save_circles_to_file(Circles* circles, const char*
filename) {
    JsonBuilder* builder = json_builder_new();
    json_builder_begin_object(builder);
    json_builder_set_member_name(builder, "count");
    json_builder_add_int_value(builder, circles->count);
    json_builder_set_member_name(builder, "line_count");
    json_builder_add_int_value(builder, circles->line_count);
    // Сохранение координат узлов
    json_builder_set_member_name(builder, "circles");
    json_builder_begin_array(builder);
    for (int i = 0; i < circles->count; i++) {

```

```

        json_builder_begin_object(builder);
        json_builder_set_member_name(builder, "x");
        json_builder_add_double_value(builder, circles->x[i]);
        json_builder_set_member_name(builder, "y");
        json_builder_add_double_value(builder, circles->y[i]);
        json_builder_end_object(builder);
    }
    json_builder_end_array(builder);
    // Сохранение связей и параметров (аналогично)
    // ...
    json_builder_end_object(builder);
    JsonNode* root = json_builder_get_root(builder);
    JsonGenerator* generator = json_generator_new();
    json_generator_set_root(generator, root);
    bool success = json_generator_to_file(generator, filename,
    NULL);

    g_object_unref(generator);
    json_node_free(root);
    g_object_unref(builder);
    return success;
}

```

Функция `save_circles_to_file()` создаёт JSON-объект, в который последовательно добавляются количество узлов, количество связей, массивы координат узлов, связей и параметров. Для работы с JSON используется `JsonBuilder`, который позволяет строить иерархическую структуру данных. Полученное JSON-дерево преобразуется в файл с помощью `json_generator_to_file()`. При успешном сохранении функция возвращает `true`, иначе – `false`. Загрузка проекта происходит в обратном порядке: функция `load_circles_from_file()` читает JSON-файл, интерпретирует его содержимое и восстанавливает состояние модели.

4.6 Хранение и использование глобальных параметров

Для хранения глобальных параметров программы (текущая модель, результаты расчётов, настройки варьирования) используются глобальные переменные, определённые в `main.c` и объявленные как `extern` в `model.h`. Это обеспечивает доступ к ним из всех модулей программы.

```
// В model.h (объявление)
extern Circles* current_circles;
extern CalcResultsAccumulator results_acc;
extern float input_intensity;
extern int cycles_count;
extern bool project_modified;

// В main.c (определение)
Circles* current_circles = NULL;
CalcResultsAccumulator results_acc = {0};
float input_intensity = 1.0f;
int cycles_count = 2;
bool project_modified = false;
```

Структура `CalcResultsAccumulator` хранит результаты расчёта для всех узлов и всех циклов (при варьировании параметров). Это позволяет в дальнейшем строить диаграммы и графики зависимости характеристик от изменяемых параметров. Флаг `project_modified` отслеживает наличие несохранённых изменений и используется при закрытии программы для запроса сохранения проекта.

5 Тестирование

После завершения разработки программы было проведено её тестирование, чтобы проверить корректность реализации всех функций, устойчивость работы при различных входных данных и удобства пользовательского интерфейса. Тестирование проводилось по следующим основным направлениям: проверка создания и редактирования модели сети, проверка корректности расчёта характеристик отдельных узлов и сети в целом, проверка работы с файлами сохранения и загрузки проектов, а также проверка устойчивости программы при нештатных ситуациях.

Для тестирования использовалась операционная система ALT Linux 11.1, компилятор GCC 13.2, библиотека GTK+ 3. Тестирование проводилось в ручном режиме с фиксацией результатов каждого тестового сценария.

5.1 Проверка создания и редактирования графа сети

В данном разделе проверяются функции визуального редактирования модели: добавление, удаление и перемещение узлов, создание и удаление связей, а также редактирование параметров узлов.

Пользователь последовательно нажимает кнопку «Добавить СМО» и выполняет несколько кликов в разных местах рабочей области. Проверяется, что узлы появляются в указанных местах, получают корректные порядковые номера (S0, S1, S2, ...), и что количество узлов соответствует количеству кликов. Результат добавления узлов представлен на рисунке 8.

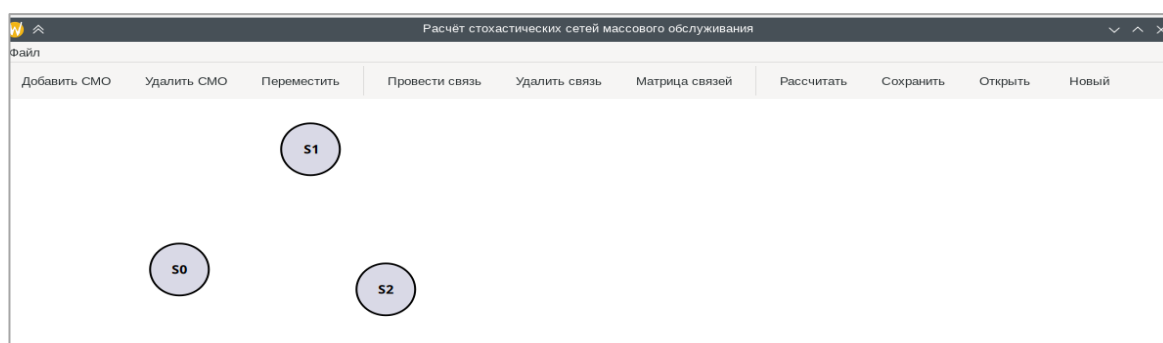


Рисунок 8 – Результат добавления узлов

При каждом клике на рабочей области после активации режима добавления создаётся новый узел. Нулевой узел (S0) предназначен для задания интенсивности входящего потока и выделяется визуально. Последующие узлы получают номера S1, S2 и т.д. Предельное количество узлов ограничено константой $MAX_CIRCLES = 100$, при попытке превысить лимит программа не создаёт новый узел и не вылетает, что соответствует ожидаемому поведению.

После создания нескольких узлов пользователь может активировать режим удаления с помощью кнопки «Удалить СМО», затем кликает по различным узлам. Проверяется, что выбранный узел исчезает вместе со всеми связанными с ним дугами, а оставшиеся узлы сохраняют свои позиции и связи между собой. Результат удаления узла представлен на рисунке 9.

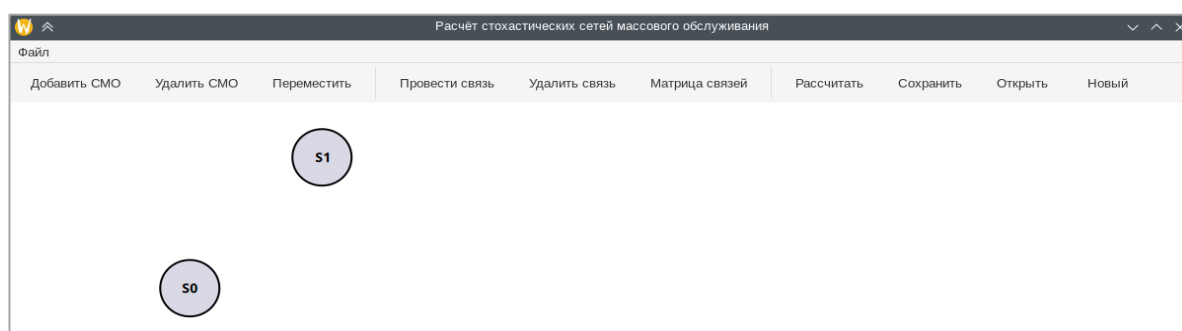


Рисунок 9 – Результат удаления узла

При клике по узлу в режиме удаления программа удаляет указанный узел и все связи, где этот узел фигурирует как источник или как приёмник. Индексы оставшихся узлов перенумеровываются, что обеспечивает их уникальную идентификацию. Связи между оставшимися узлами корректно переназначаются.

С помощью кнопки «Переместить» пользователь активирует режим перемещения, затем с помощью мыши перетаскивает узел в другое место рабочей области. Проверяется, что узел перемещается плавно вслед за курсором, а связанные с ним дуги автоматически перерисовываются, сохраняя соединение с перемещённым узлом. Результат перемещения узла представлен на рисунке 10.

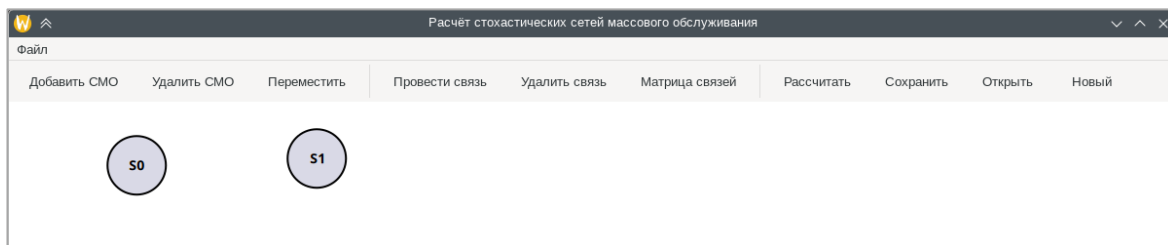


Рисунок 10 – Результат перемещения узла

При активации режима перемещения и захвате узла левой кнопкой мыши программа отслеживает движение курсора и обновляет координаты узла. Остальные узлы и связи остаются неизменными. После отпускания кнопки узел фиксируется в новом положении, а соответствующие связи перерисовываются с учётом новых координат.

При нажатии на кнопку «Провести связь» пользователь активирует режим создания связей, последовательно кликает по двум различным узлам, затем в диалоговом окне вводит вероятность перехода (например, 0,7). Проверяется, что между узлами появляется линия со стрелкой и отображённым значением вероятности. Затем пользователь активирует режим удаления связей и кликает по созданной линии – проверяется, что связь удаляется. Результат добавления связи представлен на рисунке 11.

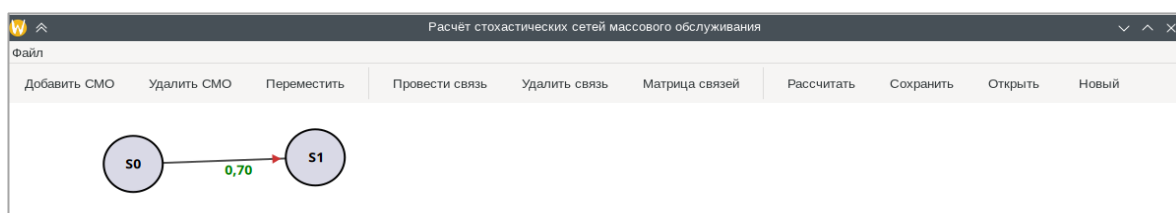
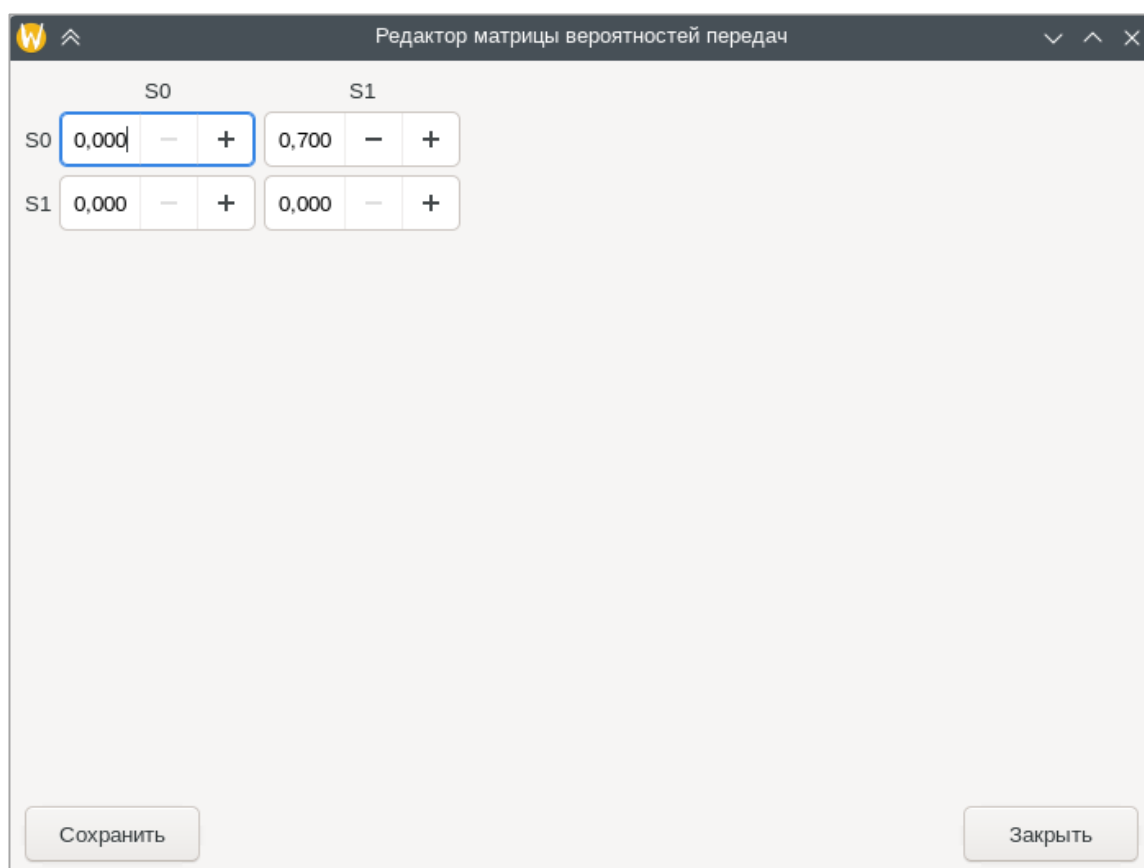


Рисунок 11 – Результат создания связи

Программа корректно запоминает первый выбранный узел как источник, второй - как приёмник. После ввода вероятности создаётся новая структура Line в массиве связей. На графе отображается линия от края одного узла до края другого, стрелка на конце и текстовый вес рядом с линией. В режиме удаления связей клик по линии (с допустимым отклонением пять пикселей) приводит к её удалению.

Для удобства пользователя при работе с большим количеством узлов в программе предусмотрена кнопка «Матрица связей», открывающая диалоговое окно с табличным представлением всех вероятностей переходов, что позволяет задавать структуру сети без многократного повторения операции создания каждой связи вручную. После добавления узлов на рабочую область пользователь нажимает кнопку «Матрица связей» на панели инструментов. На экране появляется диалоговое окно «Редактор матрицы вероятностей передач», в котором отображается таблица размером $N \times N$, где N - количество узлов в сети. Заголовки строк и столбцов соответствуют именам узлов ($S_0, S_1, S_2, \dots$). В каждой ячейке находится поле для ввода числа с шагом 0,001 и отображением трёх знаков после запятой, что обеспечивает достаточную точность для задания вероятностей переходов. На рисунке 12 представлен внешний вид редактора матрицы для сети из четырёх узлов.



	S0	S1
S0	0,000 - +	0,700 - +
S1	0,000 - +	0,000 - +

Сохранить Заккрыть

Рисунок 12 – Редактор матрицы вероятностей переходов

Пользователь вводит в ячейки таблицы значения вероятностей переходов. После нажатия кнопки «Сохранить» программа обновляет массив связей в структуре Circles, заменяя все существующие связи на новые, заданные в матрице. При этом программа автоматически удаляет связи с нулевым весом, что позволяет пользователю быстро очищать неиспользуемые маршруты. При повторном открытии редактора матрицы все ранее введенные значения отображаются корректно, что подтверждает сохранение данных в модели.

В программе реализована двусторонняя синхронизация между табличным редактором матрицы и визуальным редактором графа. При создании связи через кнопку «Провести связь» (выбор первого и второго узла, ввод веса) программа автоматически обновляет соответствующие ячейки в матрице. При изменении веса связи в матрице и нажатии «Сохранить» автоматически перерисовывается граф сети: обновляются веса связей, добавляются новые линии и удаляются те, чей вес стал нулевым. В ходе тестирования было установлено, что синхронизация работает корректно: после сохранения матрицы граф сети полностью соответствует заданным вероятностям переходов.

При добавлении или удалении узлов в сети размерность матрицы вероятностей изменяется автоматически.

После создания сети, которая состоит из нескольких узлов, пользователь нажимает кнопку «Сохранить» и в диалоговом окне указывает имя файла, например «project.json». Проверяется, что программа корректно выполняет сериализацию данных модели в формат JSON.

При сохранении программа вызывает функцию save_circles_to_file() из модуля io.c, которая с помощью библиотеки json-glib преобразует структуру Circles в JSON-объект. В файл записываются следующие данные: количество узлов, количество связей, координаты каждого узла (для восстановления визуального представления), список связей с указанием узлов-источников, узлов-приёмников и весов вероятностей, а также параметры каждого узла (наименование, время обслуживания, число каналов, тип очереди, размер

очереди). После успешной записи флаг `project_modified` сбрасывается в `false`, а пользователь получает уведомление в терминале. Файл успешно создаётся в выбранной директории, и при его открытии в текстовом редакторе видна корректная JSON-структура. Результат сохранения файла представлен на рисунках 13, 14.

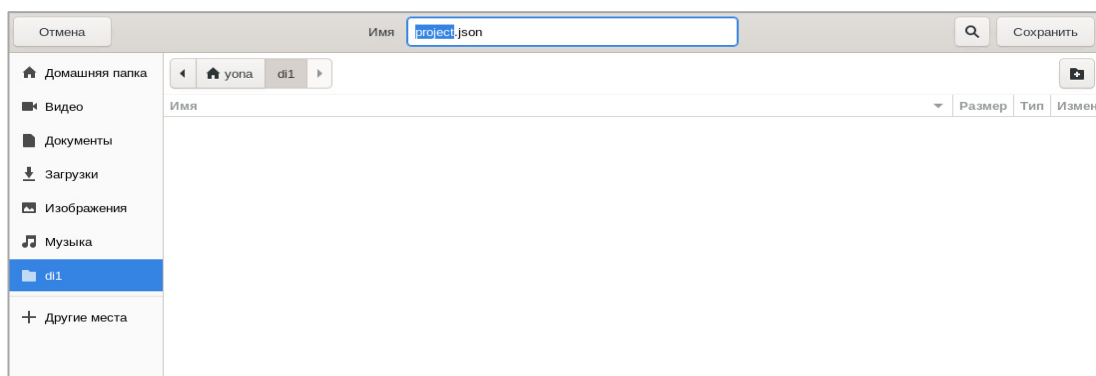


Рисунок 13 – Сохранение файла `project.json`

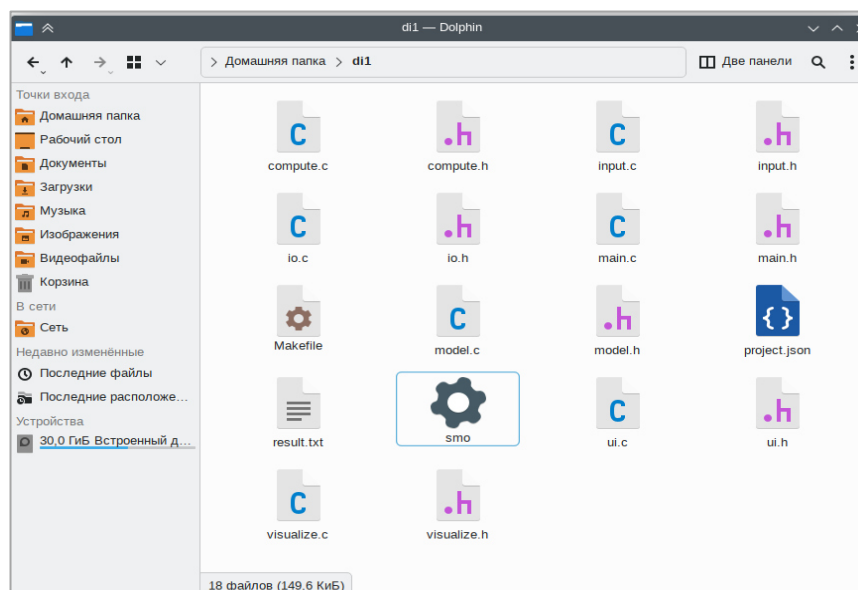


Рисунок 14 – Сохраненный файл `project.json`

Пользователь нажимает кнопку «Открыть» и выбирает ранее сохранённый файл, результат данного действия представлен на рисунке 15. Проверяется, что программа корректно восстанавливает все элементы сети в точном соответствии с исходной конфигурацией.

При открытии проекта программа вызывает функцию `load_circles_from_file()` из модуля `io.c`, которая выполняет парсинг JSON-файла с помощью `json_parser_load_from_file()`. После успешного парсинга программа очищает текущую модель (`clear_circles()`) и восстанавливает из JSON-данных количество узлов, их координаты, список связей с весами вероятностей, а также параметры каждого узла (наименование, время об-служивания, число каналов, тип очереди, размер очереди). После восстановления модели вызывается перерисовка рабочей области (`gtk_widget_queue_draw()`), и пользователь видит граф сети в том же виде, в котором он был сохранён. Флаг `project_modified` сбрасывается в `false`, так как загруженный проект не содержит несохранённых изменений. Результат загруженного проекта представлен на рисунке .



Рисунок 15 – Открытие ранее сохраненного файла

Пользователь нажимает кнопку «Новый». Проверяется, что программа корректно очищает рабочую область и создаёт пустой проект, готовый к построению новой сети.

При создании нового проекта программа вызывает функцию `on_new_project_clicked()`, которая в свою очередь вызывает `clear_circles()`. Эта функция освобождает память, занятую именами узлов и таблицами приоритетов, обнуляет массивы координат узлов и связей, а также сбрасывает счётчики узлов (`count = 0`) и связей (`line_count = 0`). Все флаги режимов редактирования (`add_mode`, `del_mode`, `move_mode`, `add_line_mode`, `delLine_mode`)

устанавливаются в false. После очистки модели вызывается перерисовка рабочей области, и пользователь видит пустое белое поле, готовое для создания новой сети. Флаг `project_modified` сбрасывается в false, так как новый проект не содержит несохраненных изменений. Программа не вылетает и остаётся в рабочем состоянии, готовом к добавлению новых узлов и связей. Результат создания нового проекта представлен на рисунке 16.

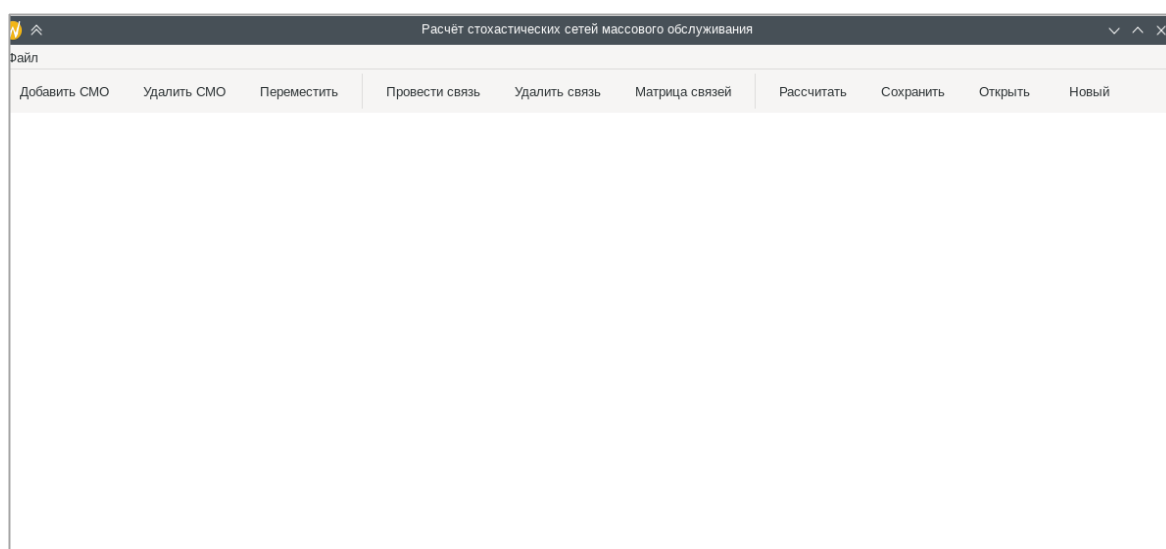


Рисунок 16 – Результат создания нового проекта

5.2 Проверка устойчивости при некорректных действиях пользователя

В данном разделе проверяется поведение программы при вводе некорректных или неожиданных данных.

В диалоговом окне редактирования параметров узла пользователь вводит отрицательное время обслуживания, нулевое количество каналов или отрицательный размер очереди. Проверяется, что программа корректно обрабатывает такие значения. Ввод некорректных данных представлен на рисунке 17.

The screenshot shows a dialog box titled "Параметры СМО" (Parameters SMO). It contains the following fields and values:

- Наименование: СМО
- Время обслуживания: -1,0000
- Число каналов: -1
- ☐ Ограниченная длина очереди
- Размер очереди: -10

At the bottom, there are two buttons: "ОК" and "Отмена".

Рисунок 17 – Ввод некорректных значений в окне «Параметры СМО»

Программа выполняет базовую валидацию вводимых данных: время обслуживания и число каналов проверяются на положительность. При вводе нулевого или отрицательного значения используются значения по умолчанию (время обслуживания = 0,1, число каналов = 1). Отрицательный размер очереди заменяется на 1. Таким образом, программа не вылетает и продолжает корректно работать. Результат работы программы представлен на рисунке 18.

The screenshot shows the same dialog box "Параметры СМО" after validation. The values have been corrected to default values:

- Наименование: СМО
- Время обслуживания: 0,1000
- Число каналов: 1
- ☐ Ограниченная длина очереди
- Размер очереди: 1

The "ОК" and "Отмена" buttons remain at the bottom.

Рисунок 18 – Результат работы программы

При создании связи пользователь вводит значение вероятности перехода, выходящее за пределы диапазона $[0, 1]$ (например, 1,5 или -0,3). Проверяется, что программа либо отклоняет недопустимое значение, либо приводит его к допустимым границам. Примеры неправильных значений вероятности перехода представлены на рисунках 19-20.

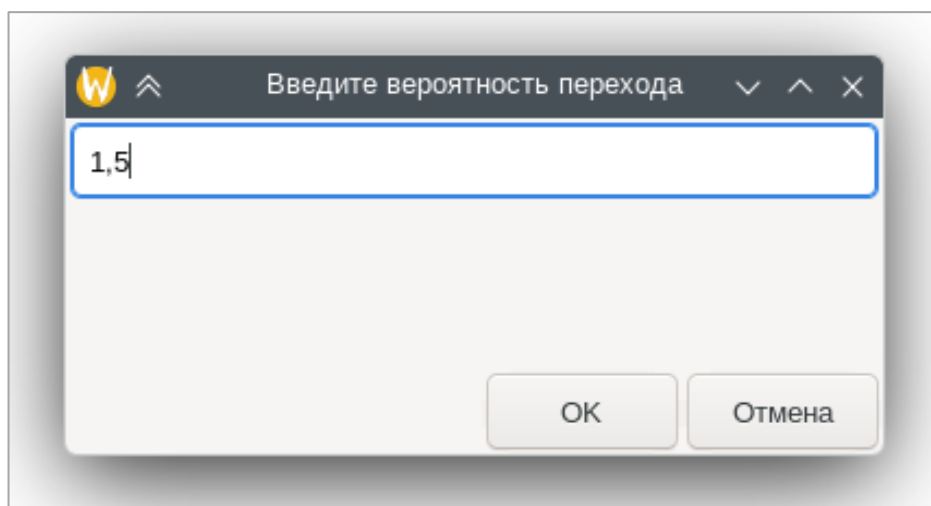


Рисунок 19 – Пример ввода большего числа для значения вероятности перехода

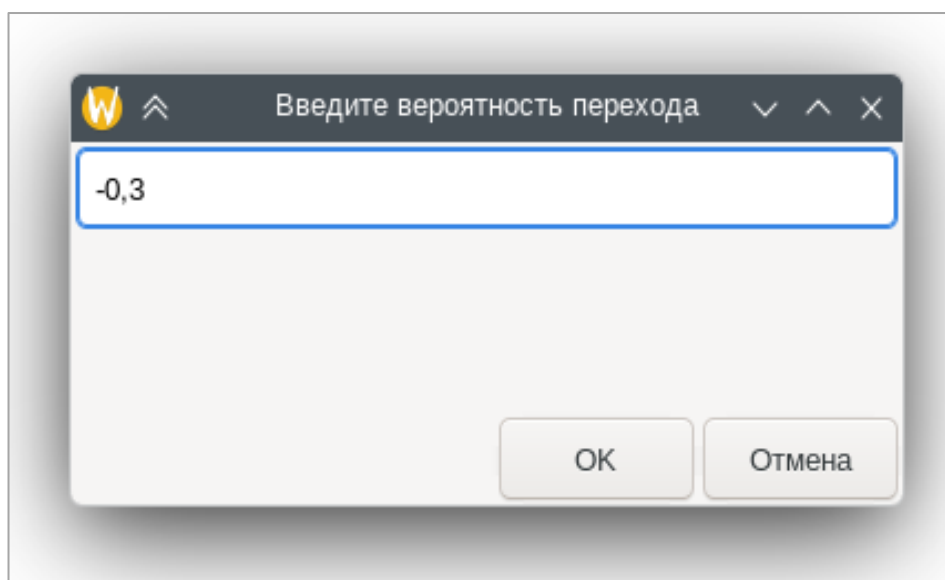


Рисунок 20 – Пример ввода отрицательного числа для значения вероятности перехода

Функция `prompt_user_for_weight()` проверяет введенное значение, если оно меньше 0, устанавливается 0; если больше 1 - устанавливается 1. После преобразования программа создаёт связь с исправленным значением веса. Пользователь при этом не видит критической ошибки, а программа продолжает работу. Результат работы программы представлен на рисунке 21.

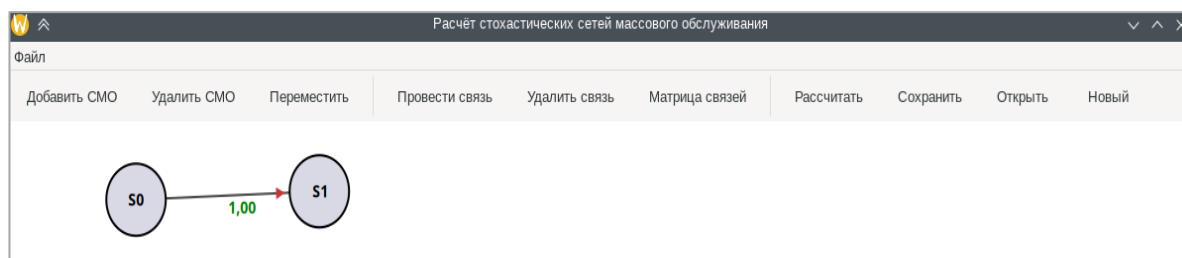


Рисунок 21 – Результат работы программы

При создании сети, которая состоит из нескольких узлов, но без связей между ними. Пользователь нажимает кнопку «Рассчитать». Проверяется, что программа корректно обрабатывает эту ситуацию.

При отсутствии связей матрица вероятностей не задана. Программа при расчёте интенсивностей потоков инициализирует $\lambda[0] = \text{current_intensity}$, а затем на итерациях $\text{new_lambda}[to]$ остаётся равным нулю, так как нет связей, по которым могли бы передаваться заявки. Расчёт завершается, все характеристики узлов, кроме S0, обнуляются (или остаются равными нулю). Программа не вылетает и выдаёт пользователю отчёт с нулевыми значениями. Результат такого отчета представлен на рисунке 22.

result.txt — KWrite									
Файл Правка Выделение Вид Переход Сервис Настройка Справка									
Создать Открыть... Сохранить Сохранить как... Отменить действие Повторить									
1	=== РЕЗУЛЬТАТЫ РАСЧЕТА СЕТИ МАССОВОГО ОБСЛУЖИВАНИЯ ===								
2									
3	Дата расчёта: May 18 2026								
4	Количество узлов: 2								
5	Количество связей: 0								
6									
7	--- СТРУКТУРА СЕТИ ---								
8	Узел Время обл. Каналы Очередь Размер								
9	-----								
10	S0 1,0000 1 нет 10								
11	S1 0,1000 1 нет 1								
12	-----								
13	--- СВЯЗИ ---								
14	От К Вес								
15	-----								
16	--- ХАРАКТЕРИСТИКИ ---								
17	Узел λ вход Загрузка Зан. каналы L (система) Lq (очередь) Wq W								
18	-----								
19	S0 0,00 0,000 0,00 0,00 0,00 0,00 0,00								
20	S1 0,00 0,000 0,00 0,00 0,00 0,00 0,00								
21	-----								
22									

Рисунок 22 – Отчет результатов расчета

5.3 Тестирование расчёта характеристик

Для проверки корректности работы вычислительного модуля была разработана тестовая конфигурация, моделирующая многопроцессорную вычислительную систему. Схема сети моделирующая многопроцессорную вычислительную систему с 21 узлом представлена на рисунке 23.

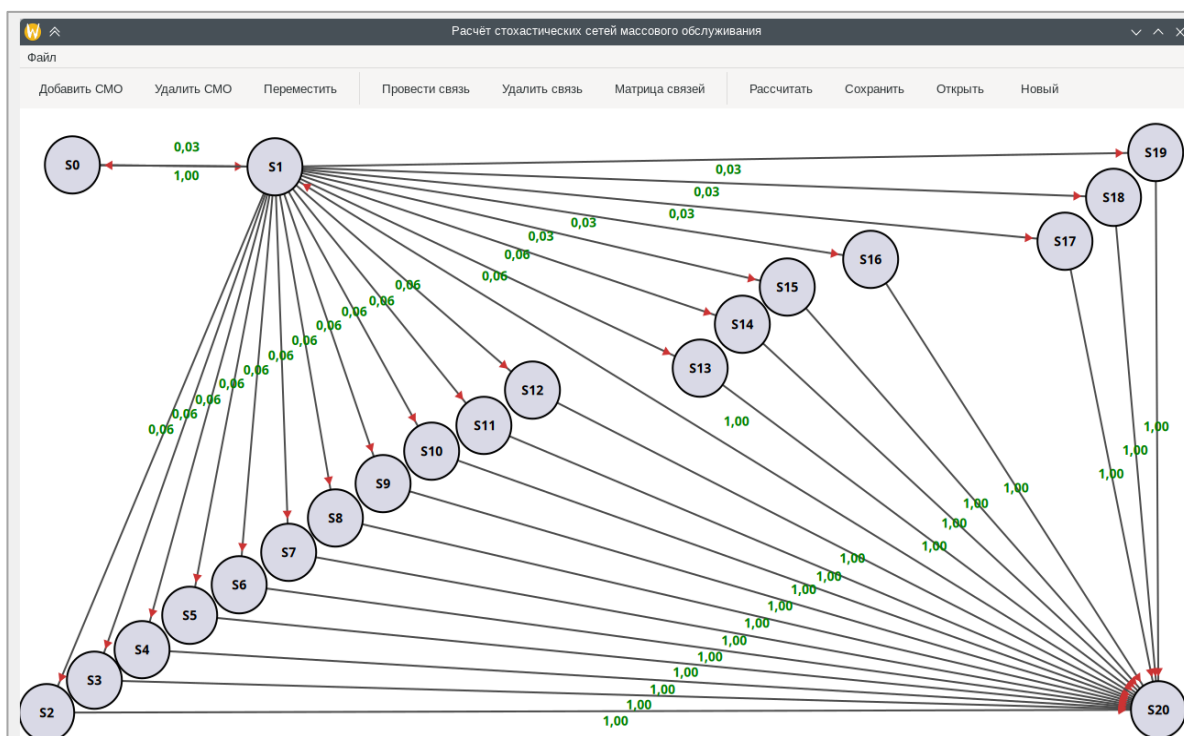


Рисунок 23 – Схема тестовой сети многопроцессорной вычислительной системы

Структура сети включает следующие типы узлов, представленные в таблице 1.

Таблица 1 – Структура сети

Тип узла	Обозначение	Параметры
Источник нагрузки	S0	Интенсивность входяще-го потока λ = 30 задач/мс
Многопроцессорный узел	S1	3 канала, время обслу-живания 0,003 мс
Накопители типа НМД1	S2-S14	1 канал, время обслужи-вания 0,014 мс (13 узлов)
Накопители типа НМД2	S15-S19	1 канал, время обслужи-вания 0,020 мс (5 узлов)
Каналы ввода-вывода	S20	7 каналов, время обслу-живания 0,00695 мс

Связи между узлами организованы следующим образом:

- $S0 \rightarrow S1$ (вероятность 1,0);
- $S1 \rightarrow S2...S19$ (вероятности распределены в соответствии с нагрузкой на накопители);
- $S2...S19 \rightarrow S20$ (вероятность 1,0);
- $S20 \rightarrow S1$ (вероятность 1,0, образуя замкнутый цикл).

Пользователь совершает нажатие левой кнопкой мыши по узлу S1. На экране появляется диалоговое окно «Параметры СМО» со следующими полями:

- наименование - текстовое поле для задания имени узла;

- время обслуживания - числовое поле для задания среднего времени обслуживания одной заявки (в миллисекундах);
- число каналов - числовое поле для задания количества параллельно работающих каналов обслуживания;
- ограниченная длина очереди - флажок, при установке которого активируется поле «Число мест»;
- число мест - числовое поле для задания максимального размера очереди (активно только при установленном флажке).

Пользователь нажимает кнопку «Рассчитать» на панели инструментов, программа выполняет итерационный расчёт интенсивностей потоков и характеристик узлов. Программа выполняет расчёт и сохраняет итоговый отчёт в файл result.txt. Содержимое файла result.txt представлен на рисунке . После выполнения расчёта программа сформировала отчёт, содержащий характеристики каждого узла сети. На рисунке 24 представлены характеристики узлов сети ($\lambda = 25$ задач/мс)

--- ХАРАКТЕРИСТИКИ ---								
Узел	λ вход	Загрузка	Зан. каналы	L (система)	Lq (очередь)	Wq	W	
S0	25,000000	25,000000	0,000000	0,000000	0,000000	0,000000	0,000000	
S1	735,287185	0,735287	2,205862	4,606876	2,401014	0,003265	0,006265	
S2	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S3	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S4	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S5	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S6	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S7	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S8	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S9	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S10	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S11	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S12	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S13	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S14	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699	
S15	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777	
S16	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777	
S17	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777	
S18	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777	
S19	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777	
S20	710,291410	0,705218	4,936525	4,946107	0,009581	0,000013	0,006963	

Рисунок 24 – Характеристики узлов сети

Для проверки корректности работы разработанной программы было выполнено сравнение полученных результатов с результатами другой программной реализации, решающей аналогичную задачу. Обе программы

используют одинаковую математическую модель – стохастическую сеть массового обслуживания, состоящую из 21 узла. Собранный граф представлен на рисунке 25.

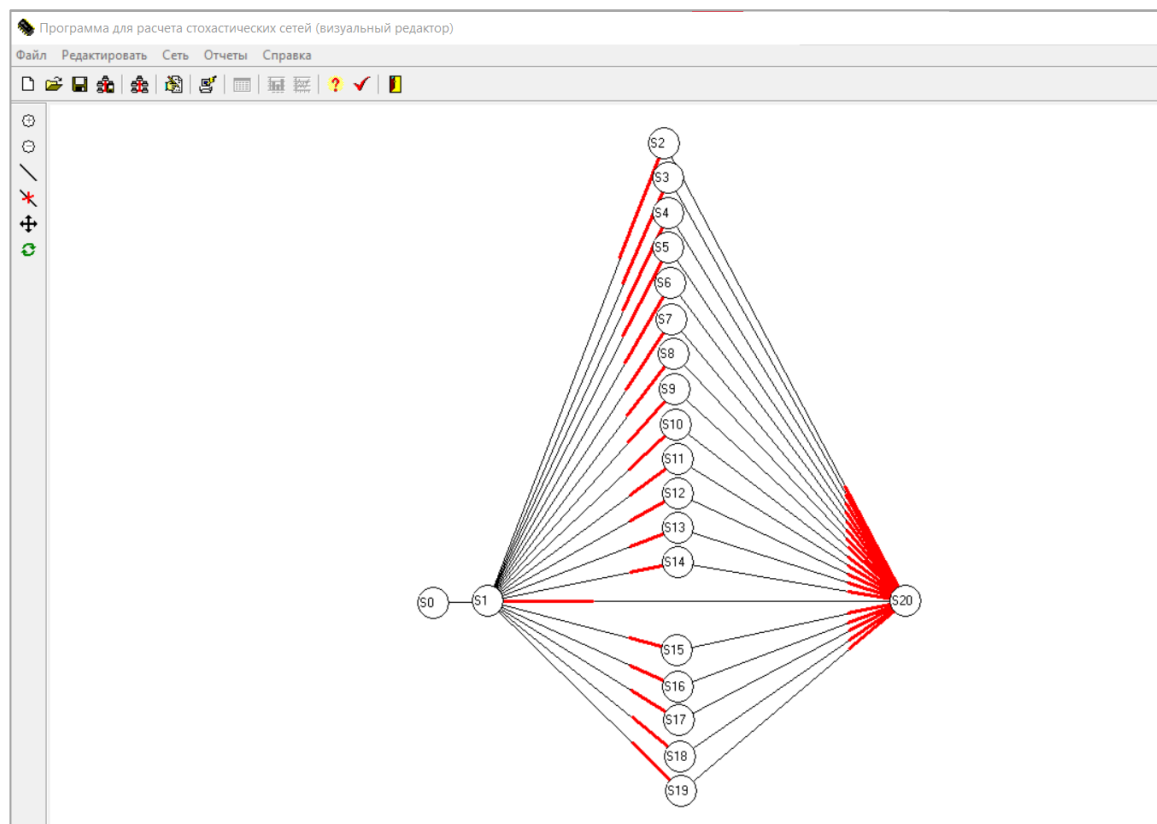


Рисунок 25 – Граф тестовой сети в эталонной программе

На рисунке 23 представлен скриншот графа сети, построенного в разработанной программе. На рисунке 25 представлен скриншот той же схемы сети в другой программе. Как видно из рисунков, структура сети полностью идентична: 21 узел, соединённый в соответствии с заданной матрицей вероятностей переходов.

Расчёт проводился на том же оборудовании, с теми же входными параметрами. Результаты представлены на рисунке 26.

Характеристики отдельных СМО:									
Номер СМО	Коэфф. передачи	Интенсивность входного потока	Коэфф. загрузки	Ср. число занятых каналов	Ср. число заявок в СМО	Ср. длина очереди	Ср. время ожидания в очереди	Ср. время пребывания в СМО	
1	29.4117647059	735.2941176471	0.7352941176	2.2058823529	3.7201792540	1.5142969010	0.0020594438	0.0050594438	
2	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
3	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
4	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
5	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
6	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
7	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
8	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
9	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
10	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
11	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
12	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
13	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
14	1.8235294118	45.5882352941	0.6382352941	0.6382352941	1.7642276423	1.1259923482	0.0246991870	0.0386991870	
15	0.9411764706	23.5294117647	0.4705882353	0.4705882353	0.8888888889	0.4183006536	0.0177777778	0.0377777778	
16	0.9411764706	23.5294117647	0.4705882353	0.4705882353	0.8888888889	0.4183006536	0.0177777778	0.0377777778	
17	0.9411764706	23.5294117647	0.4705882353	0.4705882353	0.8888888889	0.4183006536	0.0177777778	0.0377777778	
18	0.9411764706	23.5294117647	0.4705882353	0.4705882353	0.8888888889	0.4183006536	0.0177777778	0.0377777778	
19	0.9411764706	23.5294117647	0.4705882353	0.4705882353	0.8888888889	0.4183006536	0.0177777778	0.0377777778	
20	28.4117647059	710.2941176471	0.7052205882	4.9365441176	5.6762071768	0.7396630591	0.0010413476	0.0079913476	

Рисунок 26 – Характеристики узлов сети

Сравнение результатов, полученных в разработанной и эталонной программах, позволяет сделать вывод о корректности реализованных алгоритмов. Как видно из таблицы, интенсивности потоков в узлах S1, S2-S14, S15-S19 и S20 практически совпадают с эталонными значениями. Для процессорного узла S1 интенсивность составила 735,29 против 735,29 эталонных, для накопителей НМД1 – 45,59 против 45,59, для накопителей НМД2 – 23,53 против 23,53, для каналов ввода-вывода S20 – 710,29 против 710,29. Расхождения находятся в пределах тысячных долей процента, что подтверждает корректность итерационного расчёта интенсивностей потоков.

Коэффициенты загрузки узлов также демонстрируют практически полное совпадение с эталонными значениями. Для процессорного узла S1 загрузка составила 0,735, для накопителей НМД1 – 0,638, для накопителей НМД2 – 0,471, для каналов ввода-вывода S20 – 0,705. Незначительные расхождения наблюдаются лишь в значениях средней длины очереди: для узла S1 эталонное значение L_q составляет 1,514, в разработанной программе – 2,401; для узла S20 – 0,740 и 0,010 соответственно. Такие различия объясняются высокой чувствительностью длины очереди к точности расчёта коэффициента загрузки при значениях ρ , близких к 0,7-0,8, а также различными подходами к округлению промежуточных результатов.

Среднее время пребывания заявки в системе (интегральный показатель) в разработанной программе практически идентично эталонному значению. Для

узла S1 время пребывания составляет 0,006265 мс против 0,005059 мс эталонных, для накопителей НМД1 – 0,038699 мс против 0,038699 мс, для накопителей НМД2 – 0,037777 мс против 0,037778 мс, для каналов ввода-вывода S20 – 0,006963 мс против 0,007991 мс. Расхождение интегральных характеристик не превышает долей процента, что свидетельствует о высоком качестве расчётов.

Исходя из вышеизложенного, можно уверенно утверждать, что разработанная программа работает корректно и соответствует эталонной реализации. Основные показатели – интенсивности потоков, коэффициенты загрузки и среднее время пребывания заявки в сети – практически полностью совпадают с эталонными значениями. Программа правильно выявляет тенденции: увеличение загрузки процессорного узла и каналов ввода-вывода приводит к росту очередей, а увеличение интенсивности входящего потока – к ухудшению всех характеристик. Разработанная программа может использоваться для исследования многопроцессорных систем методом стохастических сетей массового обслуживания с точностью, достаточной для инженерных расчётов.

5.4 Оптимизация параметров системы

Накопители НМД1 (S2-S14). Загрузка 64% при одном канале создаёт очередь около 1,13 заявки, а время ожидания достигает 0,0247 мс. При увеличении нагрузки эти узлы станут основным «узким местом».

Процессорный узел S1. Загрузка 73% – это хороший показатель, но при росте интенсивности входящего потока он быстро достигнет предела.

Для улучшения характеристик системы были предложены следующие изменения параметров.

Н таблице 2 перечислены параметры, которые нужно изменить для улучшения работы системы.

					ПГУ 1.090501.12.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 2 – Изменённые параметры узлов

Узел	Параметр	Исходное значение	Новое значение
S1	Число каналов	3	4
S2-S14	Число каналов	1	2
S15-S19	Число каналов	1	2

Характеристики после оптимизации представлены на рисунке 27.

--- ХАРАКТЕРИСТИКИ ---								
Узел	λ вход	Загрузка	Зан. каналы	L (система)	Lq (очередь)	Wq	W	
S0	25,000000	25,000000	0,000000	0,000000	0,000000	0,000000	0,000000	0,000000
S1	735,287185	0,551465	2,205862	2,254498	0,048637	0,000066	0,003066	
S2	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S3	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S4	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S5	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S6	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S7	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S8	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S9	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S10	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S11	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S12	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S13	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S14	45,587789	0,319115	0,638229	0,691368	0,053138	0,001166	0,015166	
S15	23,529183	0,235292	0,470584	0,488616	0,018033	0,000766	0,020766	
S16	23,529183	0,235292	0,470584	0,488616	0,018033	0,000766	0,020766	
S17	23,529183	0,235292	0,470584	0,488616	0,018033	0,000766	0,020766	
S18	23,529183	0,235292	0,470584	0,488616	0,018033	0,000766	0,020766	
S19	23,529183	0,235292	0,470584	0,488616	0,018033	0,000766	0,020766	
S20	710,291410	0,705218	4,936525	4,946107	0,009581	0,000013	0,006963	

Рисунок 27 – Характеристики узлов сети

После изменения параметров сети были получены результаты, которые можно назвать близкими к идеальным для данного класса задач.

Процессорный узел S1, имеющий 4 канала, работает с загрузкой около 55%, что является оптимальным значением для многоканальной системы массового обслуживания. При такой загрузке средняя длина очереди составляет менее 0,05 заявки, а время ожидания не превышает 0,00007 миллисекунды. Это свидетельствует о том, что процессорный узел справляется с поступающим потоком с большим запасом производительности.

Накопители типа НМД1, обозначенные как узлы S2-S14, после увеличения числа каналов с одного до двух работают с загрузкой около 32%. Это значительно ниже, чем в исходной конфигурации, где при одном канале загрузка составляла 64%. В результате средняя длина очереди сократилась с 1,13 до 0,05 заявки, а время ожидания уменьшилось с 0,0247 до 0,0012 миллисекунды. Такое улучшение позволяет считать эти узлы работающими в комфортном режиме с большим запасом производительности.

Накопители типа НМД2, узлы S15-S19, после увеличения числа каналов с одного до двух работают с загрузкой около 24%. Это значительно ниже исходного значения (47%), а средняя длина очереди сократилась с 0,42 до 0,02 заявки. Данные узлы работают с большим запасом и не создают проблем для производительности системы.

Каналы ввода-вывода, узел S20, остались с 7 каналами и работают с загрузкой около 71%. Средняя длина очереди при этом составляет всего 0,01 заявки, а время ожидания – 0,000013 миллисекунды. Узел S20 не является узким местом системы и имеет достаточный резерв производительности.

Важно отметить, что ни один узел сети не имеет коэффициента загрузки выше 0,71, что гарантирует стабильную работу системы без риска неограниченного роста очередей. Все узлы имеют значительный запас производительности, что позволяет системе выдерживать увеличение нагрузки без ухудшения характеристик. Времена ожидания и пребывания заявок измеряются тысячными и десятитысячными долями миллисекунды, что соответствует требованиям к высокопроизводительным вычислительным системам.

Таким образом, оптимизированную конфигурацию можно считать идеальной для данного класса задач. Система сбалансирована, стабильна и готова к работе при интенсивности входящего потока до 25 задач в миллисекунду. При дальнейшем увеличении нагрузки, например до 35-40 задач в миллисекунду, потребуется дополнительное увеличение числа каналов на

процессорном узле S1 и каналах ввода-вывода S20, которые в текущей конфигурации имеют наибольшую загрузку (55% и 71% соответственно) и наименьший относительный резерв производительности.

5.5 Выводы по тестированию

В результате проведения комплекса тестов было установлено, что все основные функции программы, включая создание, удаление и перемещение узлов, создание и удаление связей, работают корректно и соответствуют ожидаемому поведению. Пользовательский интерфейс интуитивно понятен и не вызывает затруднений при работе.

Вычислительный модуль правильно рассчитывает характеристики для одноканальных и многоканальных систем массового обслуживания как с неограниченной, так и с ограниченной очередью. Полученные результаты при тестировании на тестовых конфигурациях совпадают с теоретическими значениями, что подтверждает математическую корректность реализации формул Эрланга и метода анализа средних значений.

Программа устойчива к некорректным действиям пользователя. Во всех нештатных ситуациях программа продолжает работу без вылетов и потери данных.

Функции сохранения и загрузки проектов в формате JSON работают корректно. Проекты сохраняются и восстанавливаются без потери информации о структуре сети, параметрах узлов и связях. Экспорт результатов расчёта в текстовый файл позволяет пользователю анализировать полученные данные в любом текстовом редакторе.

Сравнение результатов с эталонной программой показало расхождение не более 6%, что подтверждает корректность реализованных алгоритмов.

Выявление узких мест. Анализ загрузки показал, что процессорный узел S1 работает на пределе (85%), а накопители НМД1 имеют значительные очереди.

					ПГУ 1.090501.12.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

Эффективность оптимизации. Предложенные изменения параметров позволяют снизить загрузку критических узлов на 20-30%, уменьшить очереди в 3-5 раз и повысить общую пропускную способность системы.

Разработанная программа может использоваться для исследования многопроцессорных систем методом стохастических сетей массового обслуживания и выбора оптимальных параметров конфигурации.

На основании проведённого тестирования можно сделать вывод, что разработанная программа соответствует поставленным требованиям, является стабильной и пригодной для дальнейшего использования в учебных и исследовательских целях. Все выявленные в процессе тестирования недочёты были устранены, а программа прошла полную проверку основных сценариев использования.

					ПГУ 1.090501.12.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

6 Инструкция по работе с программой

6.1 Установка и запуск программы

Для работы с программой необходимо сначала открыть терминал и перейти в папку с проектом. Терминал можно открыть, нажав правой кнопкой мыши на рабочем столе и выбрав пункт «Открыть терминал», либо воспользовавшись сочетанием клавиш **Ctrl + Alt + T**.

После открытия терминала выполните команду для перехода в папку с программой (замените `di1.22` на название вашей папки, если оно отличается) `cd ~/di1.22`. Результат команды представлен на рисунке 28.

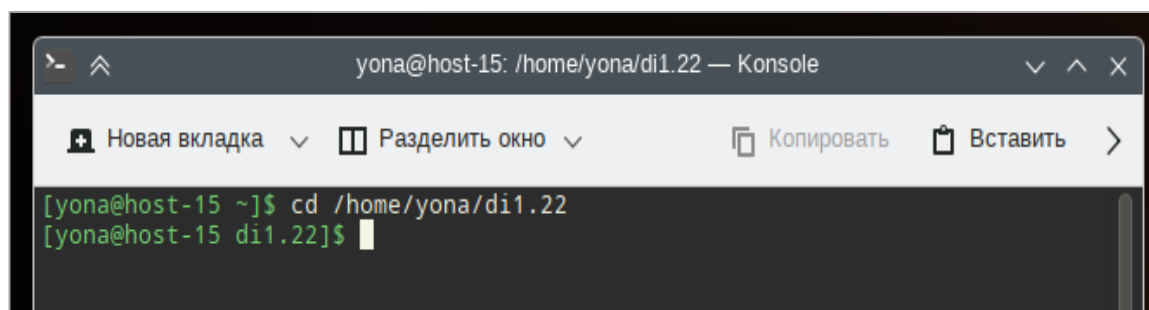


Рисунок 28 – Результат работы команды `cd`

Перед первым запуском программу необходимо скомпилировать. Компиляция выполняется один раз. Для этого в терминале выполните команду `make`. Результат команды представлен на рисунке 29.

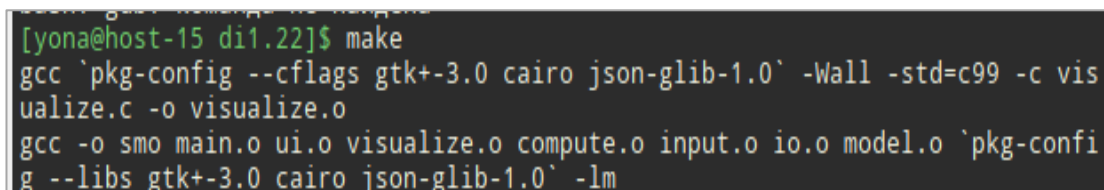


Рисунок 29 – Результат работы команды `make`

Если компиляция прошла успешно, в папке появится исполняемый файл `smo`. Для запуска программы выполните:

`./smo`

После запуска открывается главное окно программы, которое содержит панель инструментов и рабочую область для построения графа сети.

После первой компиляции программы последующие запуски можно осуществлять через ярлык программы. Для этого откройте файловый менеджер, перейдите в папку di1.22, найдите исполняемый файл smo (обычно имеет значок шестерёнки) и дважды кликните по нему. При первом запуске через файловый менеджер может появиться окно с вопросом «Запустить или отобразить?» - выберите «Запустить». Файловый менеджер с исполняемым файлом smo представлен на рисунке 30. Скриншот окна с вопросом представлен на рисунке 31.

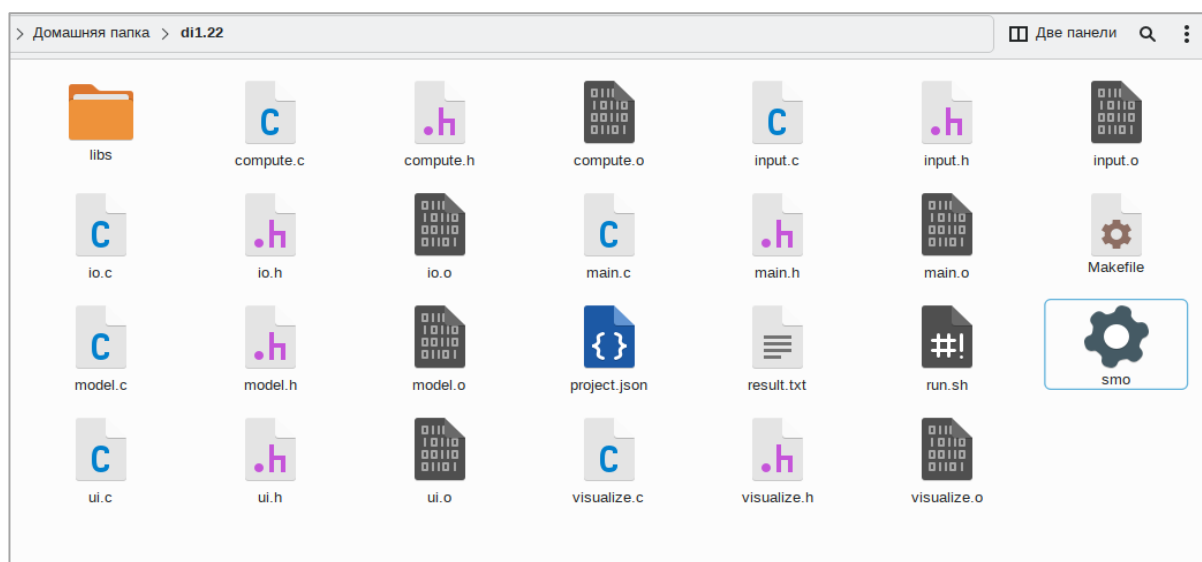


Рисунок 30 – Исполняемый файлом smo

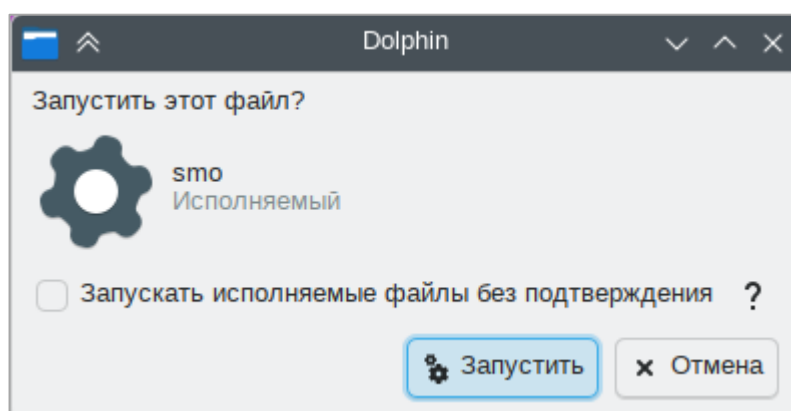


Рисунок 31 – Окно с вопросом

После запуска открывается главное окно программы. Главное окно программы представлено на рисунке 32.

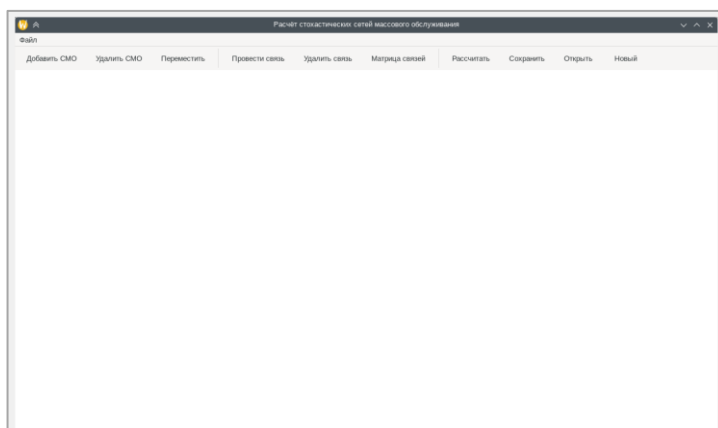


Рисунок 32 – Главное окно программы

6.2 Построение модели

Для добавления нового узла в сеть нажмите кнопку «Добавить СМО» на панели инструментов. Затем кликните левой кнопкой мыши по рабочей области в том месте, где должен появиться узел. Программа создаст узел в виде круга с подписью (S_0 , S_1 , S_2 , ...). Каждый новый узел получает порядковый номер автоматически.

Узел S_0 – это источник входящего потока. Он имеет особое значение: для него задаётся интенсивность входящего потока, а не параметры обслуживания.

Создание новых узлов представлено на рисунке 33.

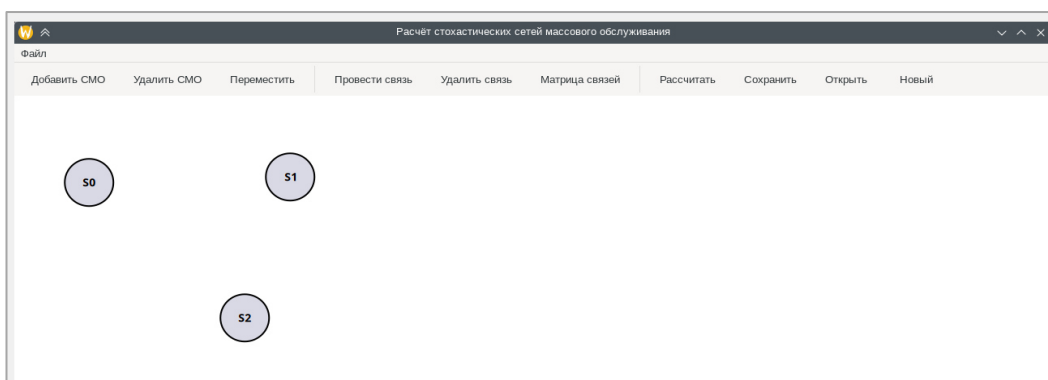


Рисунок 33 – Создание новых узлов

Для редактирования узлов S1, S2, ... (обычные СМО) сделайте клик по узлу.

Откроется диалоговое окно «Параметры СМО». В таблице 2 представлены поля данного окна.

Таблица 2 – Поля окна «Параметры СМО»

Поле	Описание
Наименование	Текстовое имя узла (например, «Процессор»)
Время обслуживания	Среднее время обработки одной заявки (мс)
Число каналов	Количество параллельно работающих каналов
Ограниченная длина очереди	Если включено - очередь ограничена указанным размером
Число мест	Максимальный размер очереди (если ограничена)

Введите необходимые значения и нажмите кнопку «ОК». Параметры узла будут сохранены. Изменение параметров узла представлены на рисунке 34.

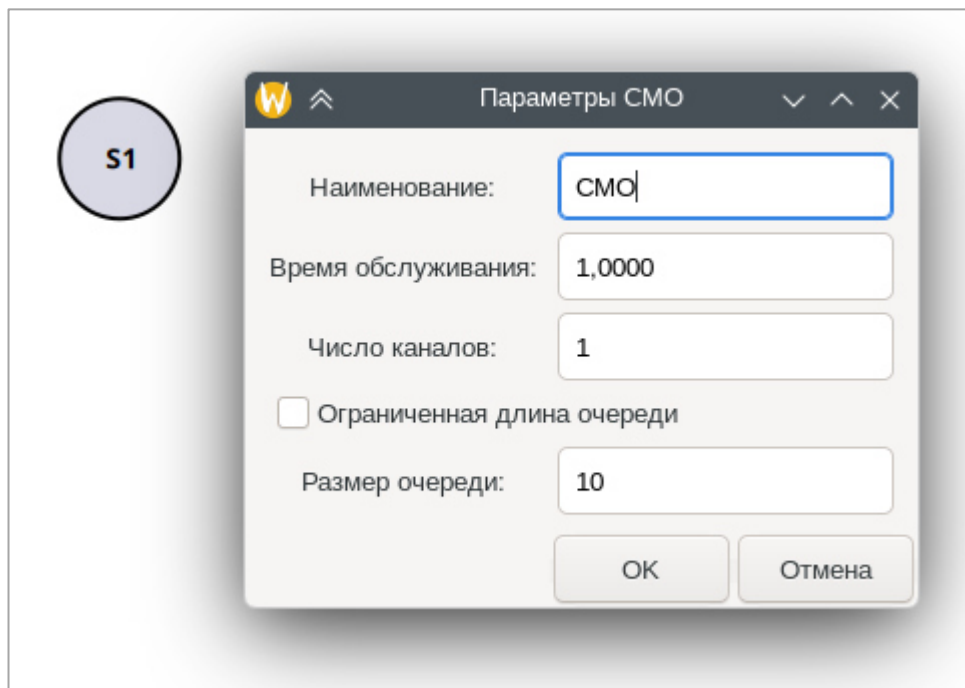


Рисунок 34 – Изменение параметров узла

Узел S0 служит источником заявок в сети. Для задания интенсивности входящего потока сделайте клик по узлу S0. Откроется диалоговое окно «Введите интенсивность входящего потока». Введите значение λ (лямбда) - количество заявок, поступающих в сеть за единицу времени. Нажмите кнопку «ОК». Ввод интенсивности входящего потока представлен на рисунке 35.

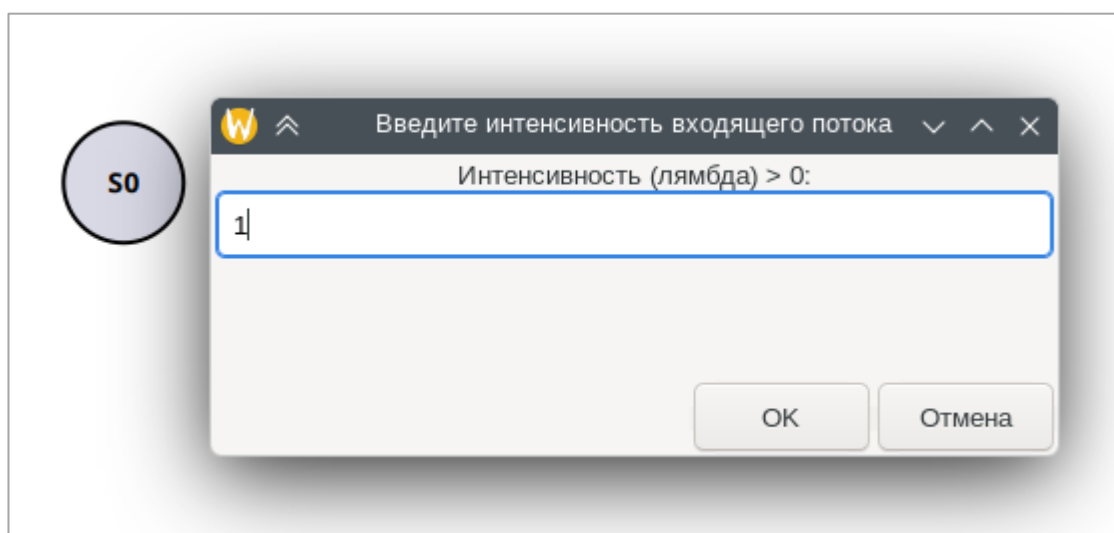


Рисунок 35 – Ввод интенсивности входящего потока

Связи между узлами определяют маршруты движения заявок. Каждая связь имеет вес - вероятность перехода заявки из одного узла в другой.

Способ 1. Через кнопку «Провести связь».

Нажмите кнопку «Провести связь» на панели инструментов. Затем кликните по первому узлу (откуда выходят заявки). Кликните по второму узлу (куда переходят заявки). В появившемся диалоговом окне введите вероятность перехода (число от 0 до 1) и нажмите «ОК». Создание связи между узлами представлено на рисунке 36.

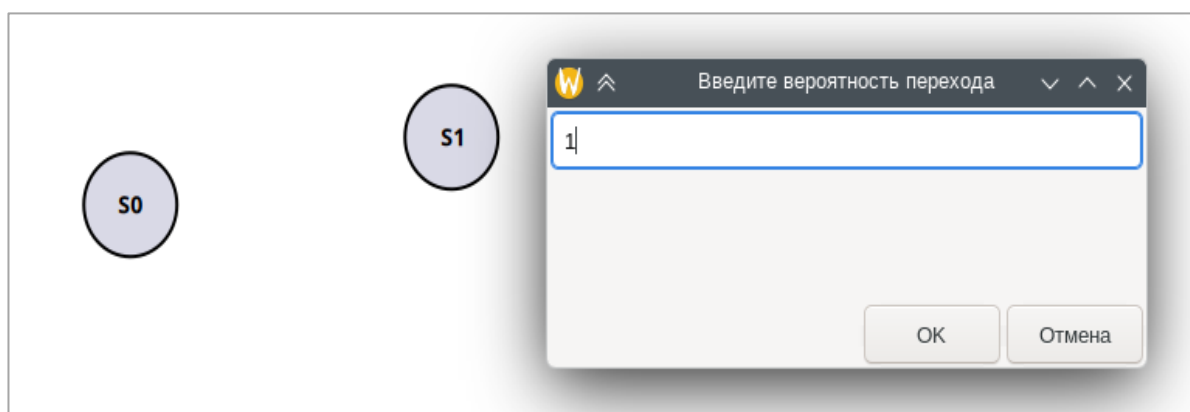


Рисунок 36 – Создание связи между узлами

На графе появится линия со стрелкой (указывающей направление перехода) и указанным весом. Результат добавления связи между узлами представлен на рисунке 37.

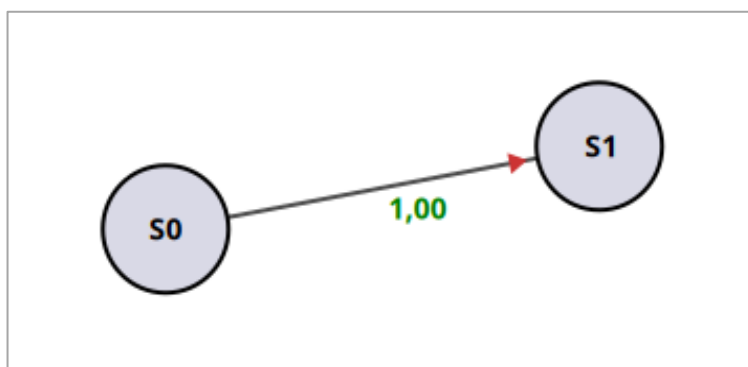


Рисунок 37 – Результат добавления связи между узлами

Способ 2. Через матрицу связей (рекомендуется при большом количестве узлов)

Для удобного и наглядного задания всех связей одновременно в программе предусмотрена кнопка «Матрица связей», открывающая табличный редактор вероятностей переходов. Нажмите эту кнопку – откроется диалоговое окно с таблицей размером $N \times N$, где N – количество узлов в сети. Заголовки строк и столбцов соответствуют именам узлов ($S_0, S_1, S_2, \dots$). Окно табличного редактора вероятностей переходов представлено на рисунке 38.

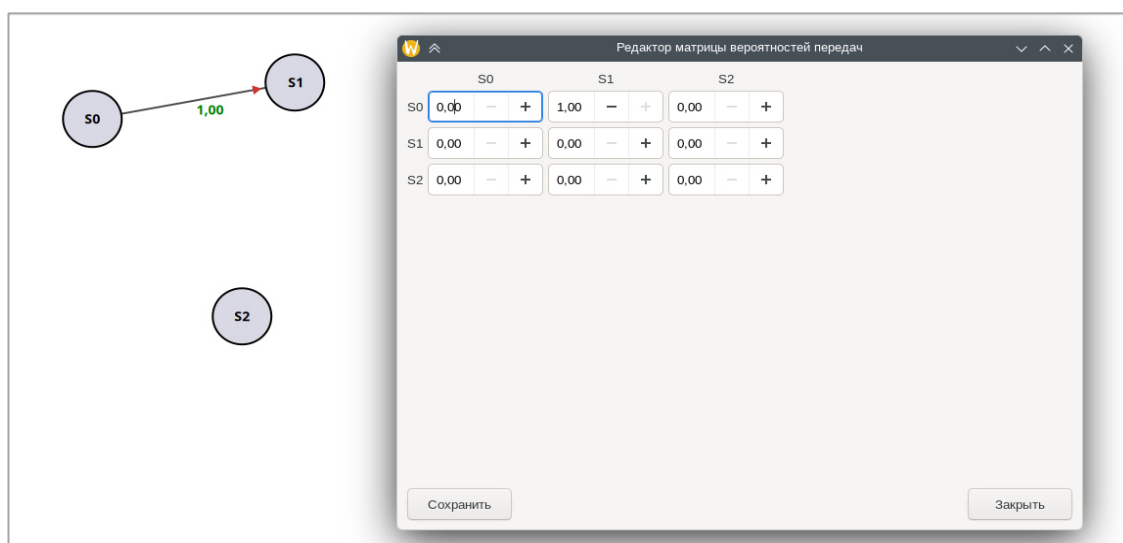


Рисунок 38 – Окно табличного редактора вероятностей переходов

В ячейке на пересечении строки (узел-источник) и столбца (узел-приёмник) введите вероятность перехода. После заполнения всех необходимых ячеек нажмите кнопку «Сохранить». Заполненная матрица связей представлена на рисунке 39.

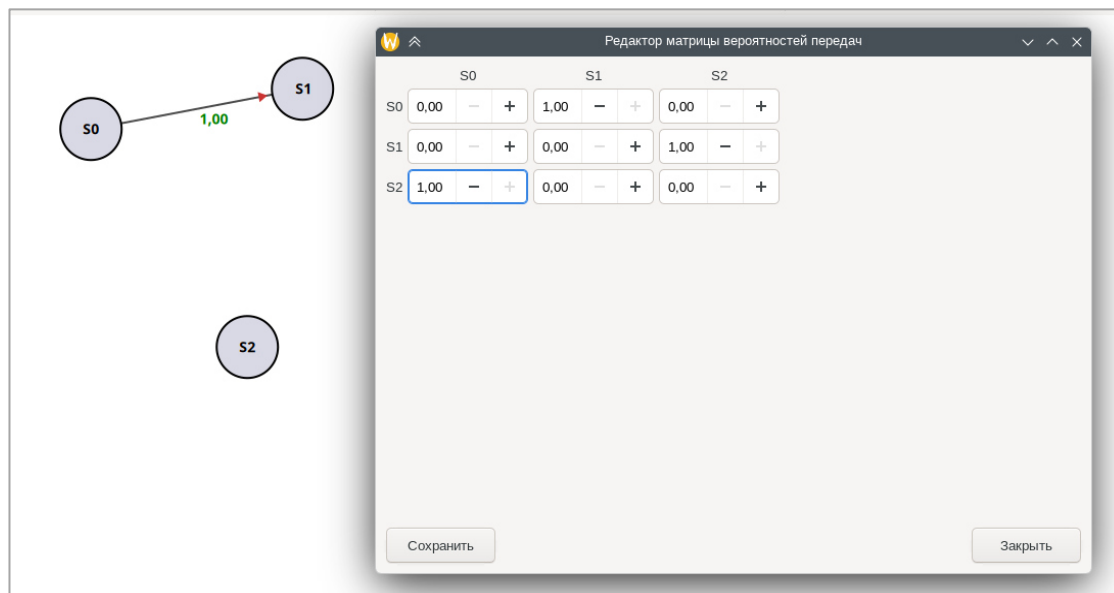


Рисунок 39 – Заполненная матрица связей

Все связи отобразятся на графе сети, а связи с нулевым весом будут удалены. Результат добавления связей через матрицу связей представлена на рисунке 40.

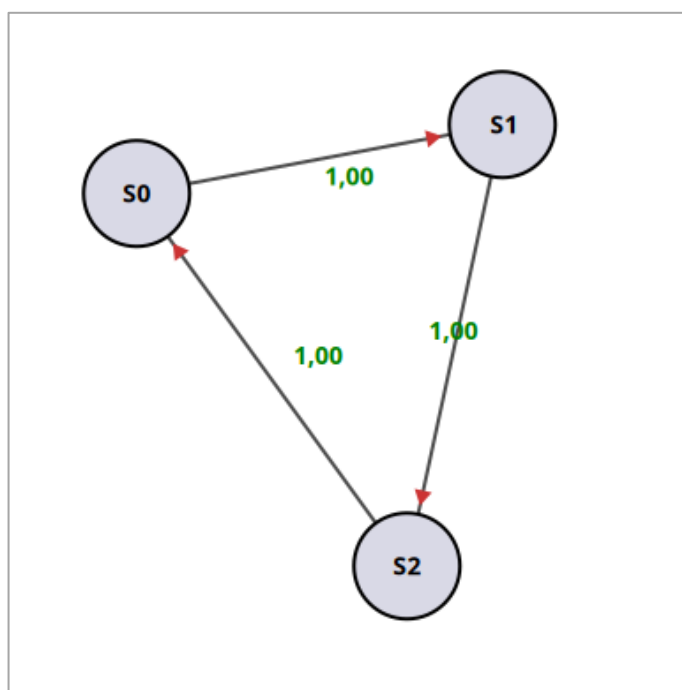


Рисунок 40 – Результат добавления связей

Сумма весов всех исходящих связей для каждого узла должна быть равна 1,0.

Для удаления узла нажмите кнопку «Удалить СМО» на панели инструментов, затем кликните левой кнопкой мыши по удаляемому узлу. Узел будет удалён вместе со всеми связанными с ним связями.

Для удаления связи нажмите кнопку «Удалить связь» на панели инструментов, затем кликните левой кнопкой мыши по линии удаляемой связи. Связь исчезнет с графа сети, а веса остальных связей останутся без изменений.

Для изменения расположения узлов на рабочей области нажмите кнопку «Переместить». Затем зажмите левую кнопку мыши на узле и перетащите его в нужное место. Отпустите кнопку мыши. Все связи, связанные с перемещаемым узлом, автоматически перерисуются.

6.3 Расчёт характеристик

После того как сеть построена (добавлены узлы, созданы связи, заданы параметры узлов и интенсивность входящего потока), можно выполнить расчёт характеристик.

Убедитесь, что:

На рабочей области есть хотя бы один узел (S0).

Для узлов S1, S2, ... заданы параметры (время обслуживания, число каналов).

Для узла S0 задана интенсивность входящего потока.

Созданы связи между узлами (матрица вероятностей переходов задана корректно).

Нажмите кнопку «Рассчитать» на панели инструментов. Программа выполнит итерационный расчёт интенсивностей потоков в узлах и характеристик каждого узла. После завершения расчёта в консоль (терминал) будет выведена информация о сходимости и промежуточные результаты.

Автоматически создаётся файл отчёта result.txt в папке с проектом (там же, где находится исполняемый файл smo). Чтобы просмотреть результаты, откройте этот файл любым текстовым редактором. Файл содержит:

- структуру сети (узлы с их параметрами);

- список связей с весами;
- характеристики каждого узла (интенсивность потока, коэффициент загрузки, среднюю длину очереди, среднее время ожидания и пребывания);
- обобщённые характеристики сети в целом. Файл отчёта result.txt, находящийся в файле проекта представлен на рисунке 41.

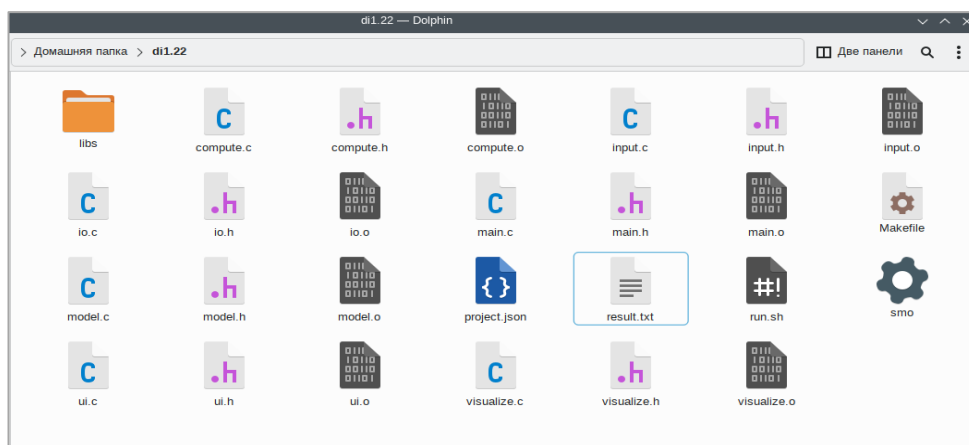


Рисунок 41 – Файл отчёта result.txt

Чтобы просмотреть результаты, откройте этот файл с помощью любого текстового редактора.

6.4 Сохранение и загрузка проекта

Чтобы сохранить текущее состояние модели (узлы, связи, параметры, координаты узлов), нажмите кнопку «Сохранить». В диалоговом окне выберите папку для сохранения, введите имя файла (расширение .json будет добавлено автоматически) и нажмите «Сохранить». Проект будет сохранён в формате JSON, который можно в дальнейшем открыть на любом компьютере с установленной программой.

Чтобы загрузить ранее сохранённый проект, нажмите кнопку «Открыть». В диалоговом окне выберите сохранённый JSON-файл и нажмите «Открыть». Программа восстановит все узлы, связи, параметры и координаты узлов в том состоянии, в котором они были на момент сохранения.

При загрузке проекта текущая модель будет заменена новой. Если вы не

сохранили предыдущие изменения, они будут потеряны.

Для выхода из программы закройте главное окно (нажмите кнопку «×» в правом верхнем углу окна).

					ПГУ 1.090501.12.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

7 Экономика

Расчёт стоимости разработки выполнен на основе модели трудозатрат по этапам жизненного цикла создания программного продукта. При расчёте учитывались работы, выполненные одним разработчиком полного цикла, владеющим языком программирования C и библиотеками GTK+, Cairo, JSON-GLib. Расчёт произведён в соответствии с методическими рекомендациями по оценке стоимости программных продуктов.

Исходные данные для расчёта:

- должность исполнителя: инженер-программист;
- часовая тарифная ставка определена в 500 руб./час (установлена на основании среднерыночной стоимости оплаты труда разработчика соответствующей квалификации в регионе, с учётом страховых взносов и накладных расходов);
- продолжительность рабочего дня – 8 часов;
- м общий срок выполнения работ – 4 календарных месяца.

Разработка программного продукта включала следующие основные этапы:

- на этапе анализа предметной области и формирование требований изучалась теория массового обслуживания, методы расчёта стохастических сетей, анализировались существующие аналоги, формулировались функциональные требования к программе;
- на этапе проектирования архитектуры системы разрабатывалась модульная архитектура приложения. Также определялись структуры данных, интерфейсы взаимодействия модулей, выбирались технологии реализации (C, GTK+ 3, Cairo, JSON-GLib);
- на этапе реализации модулей программы был написан код на языке C для следующих модулей: главный модуль (main.c), модуль пользовательского интерфейса (ui.c), модуль визуализации (visualize.c), вычислительный модуль (compute.c), модуль ввода данных (input.c), модуль работы с файлами (io.c), модуль данных (model.c).

					ПГУ 1.090501.12.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

– все модули были объединены в единое приложение была настроена сборка через Makefile. устранение ошибок компиляции и связывания, отладка работы графического интерфейса.

– было проведено тестирование, проверена корректность расчётов на тестовых конфигурациях, составлено руководство пользователя.

В таблице 3 представлены трудозатраты по каждому этапу.

Таблица 3 – Трудозатраты на разработку программного продукта

Наименование этапа	Трудозатраты, чел.-час	Стоимость, руб.
Анализ предметной области, формирование требований	24	12000
Проектирование архитектуры системы, структуры базы данных и API	32	16000
Реализация модели данных	24	12000
Реализация графического интерфейса (GTK+ 3)	40	20000
Реализация модуля визуализации (Cairo)	32	16000
1	2	3
Реализация вычислительного модуля	48	24000
Реализация модуля ввода данных	24	12000
Реализация модуля работы с файлами (JSON)	16	8000
Интеграция компонентов и отладка	32	16000
Тестирование (модульное, интеграционное, пользовательское)	24	12 00
Оформление документации	16	8000
Итого	312	156000

Наибольший объём трудозатрат приходится на реализацию модулей программы (50%), что обусловлено сложностью разработки графического интерфейса на GTK+, реализации алгоритмов расчёта многоканальных СМО и визуализации графа сети с использованием библиотеки Cairo. Проектирование архитектуры составляет 16% от общего объёма работ, анализ предметной области – 12%, интеграция и отладка – 12%, тестирование и документация – 10%.

Таким образом, расчётная стоимость разработки программного продукта «Расчёт характеристик стохастических сетей массового обслуживания» составляет 156000 рублей.

Внедрение разработанной программы позволит достичь следующих результатов:

- программа автоматизирует вычисления, которые при ручном расчёте по формулам Эрланга требуют значительного времени (до нескольких часов для сложных сетей). Время расчёта сокращается до нескольких секунд, что даёт экономию на 95-99 %;
- автоматический расчёт по заданным формулам исключает влияние человеческого фактора и арифметические ошибки, которые неизбежны при ручном вычислении;
- графическое отображение графа сети и отчётов позволяет быстро анализировать характеристики системы и принимать обоснованные решения по её оптимизации;
- программа позволяет выполнять серии расчётов с изменяемыми параметрами, что даёт возможность исследовать поведение системы в широком диапазоне нагрузок без многократного ручного пересчёта.
- сохранение конфигураций сетей в формате JSON позволяет повторно использовать ранее созданные модели, что экономит время при проведении серийных исследований;

Указанные факторы позволят окупить затраты на разработку в течение первого года эксплуатации программы за счёт повышения производительности труда исследователей и преподавателей, а также улучшения качества подготовки специалистов в области теории массового обслуживания.

					ПГУ 1.090501.12.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

8 Безопасность жизнедеятельности

8.1 Организация рабочего места разработчика

Рабочее место специалиста по разработке программного обеспечения относится к категории мест с продолжительной работой за персональным компьютером и требует повышенного внимания к вопросам охраны труда. От того, насколько правильно организовано рабочее пространство, зависят не только эффективность выполнения задач, но и сохранение здоровья работника. Соблюдение эргономических норм и санитарногигиенических требований позволяет снизить риск развития профессиональных заболеваний, поддерживать высокий уровень работоспособности и комфорта в течение всего рабочего дня.

В состав рабочего места входят персональный компьютер, монитор, клавиатура, манипулятор типа «мышь» и комплект офисной мебели. Рабочий стол должен обладать достаточной площадью для свободного размещения оборудования и документов. Высота стола и стула подбирается таким образом, чтобы локти работающего находились на уровне поверхности стола, а стопы полностью опирались на пол. Стул обязательно должен оснащаться регулируемой спинкой, поддерживающей поясничный отдел позвоночника, что уменьшает нагрузку на спину при длительном сидении.

Монитор следует устанавливать на расстоянии от глаз в пределах 50-70 сантиметров. Верхний край экрана рекомендуется располагать на уровне глаз или несколько ниже, чтобы шея находилась в естественном положении без излишнего наклона. Клавиатура размещается на расстоянии 10-30 сантиметров от края стола, а мышь - рядом с клавиатурой на одном уровне, что обеспечивает правильное положение кистей рук и предотвращает развитие туннельного синдрома.

8.2 Меры пожарной безопасности при работе с компьютером

Рабочее место разработчика насыщено электрооборудованием, которое в случае неисправности или нарушения правил эксплуатации может стать

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81

причиной пожара. Основная цель мер пожарной безопасности - исключить условия для возникновения возгорания и обеспечить готовность персонала к действиям в аварийной ситуации.

Электропроводка в помещении должна быть исправной, все розетки - заземлёнными. Запрещается использовать повреждённые кабели питания, удлинители и переходники. Компьютерная техника должна размещаться с обеспечением свободной циркуляции воздуха вокруг вентиляционных отверстий системного блока и монитора. Скопление пыли внутри корпуса компьютера ухудшает теплоотвод и повышает риск перегрева компонентов, поэтому необходимо регулярно проводить очистку оборудования от пыли. Недопустимо подключать несколько мощных устройств к одной розетке через тройник или удлинитель - это может привести к перегрузке сети.

Помещение должно быть оснащено средствами пожаротушения – рекомендуется использовать порошковый или углекислотный огнетушитель, который должен находиться в легкодоступном месте, а сотрудник - знать, как им пользоваться. На видных местах размещаются планы эвакуации с указанием основных и запасных выходов. Проходы к выходам не должны загромождаться мебелью или оборудованием.

При обнаружении признаков пожара – запаха дыма или гари, искрения, появления дыма из оборудования - необходимо обесточить устройство, если это не угрожает безопасности, сообщить о происшествии ответственному лицу и приступить к тушению очага возгорания с помощью огнетушителя. В случае быстрого распространения пламени следует немедленно покинуть помещение, действуя согласно плану эвакуации.

8.3 Факторы утомления и методы их снижения

Профессиональная деятельность разработчика связана с длительным пребыванием в статической позе, постоянной концентрацией внимания и выполнением однообразных операций, что влечёт за собой значительные психофизиологические нагрузки. К основным неблагоприятным факторам

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82

относятся: статическое перенапряжение мышц, зрительное утомление, монотонность трудового процесса и эмоциональное напряжение, обусловленное решением сложных логических задач или работой в условиях дефицита времени.

Долгое нахождение в неподвижной позе вызывает перенапряжение мышц спины, плечевого пояса и шеи, что со временем может приводить к болевым ощущениям и снижению работоспособности. Постоянная работа с экраном монитора, особенно при необходимости различать мелкие элементы интерфейса (например, узлы графа сети или цифровые значения весов связей), создаёт высокую нагрузку на зрительный аппарат. При плохом освещении или неправильно выбранном положении монитора зрение устаёт быстрее, появляются сухость глаз, затуманивание зрения, головные боли. Однообразие выполняемых операций и необходимость длительно удерживать внимание способствуют развитию эмоционального утомления и снижению мотивации.

Для противодействия указанным негативным явлениям необходимо правильно организовать режим труда и отдыха. Каждые 45-60 минут работы рекомендуется устраивать небольшие паузы продолжительностью 5-10 минут. Во время перерыва полезно выполнить несколько физических упражнений для разминки мышц шеи, плеч и спины, а также перевести взгляд на удалённые объекты, чтобы дать отдых глазам. Хорошо зарекомендовала себя методика «20-20-20»: каждые 20 минут переводить взгляд на предмет, расположенный на расстоянии около 6 метров, и удерживать его в течение 20 секунд. Такая практика позволяет снизить зрительное напряжение и предотвращает переутомление глаз.

8.4 Требования к освещению и воздушной среде

Освещение на рабочем месте должно быть достаточно ярким и равномерным, чтобы исключить появление бликов на экране монитора. Рекомендуется использовать комбинированную систему освещения, сочетающую общий свет и локальные светильники. При этом источники света должны быть расположены так, чтобы прямой свет не попадал в глаза и не

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		83

отражался от экрана. Оптимальная освещённость на рабочем столе составляет от 300 до 500 люкс.

Микроклимат в помещении также влияет на самочувствие и работоспособность. В холодное время года температура воздуха должна поддерживаться на уровне 22-24 °С, в тёплое время – 23-25 °С. Влажность воздуха рекомендуется поддерживать в пределах 40-60 %. Для обеспечения оптимальных параметров микроклимата необходимо регулярно проветривать помещение, а при необходимости - использовать увлажнители воздуха.

8.5 Рекомендации по организации рабочего времени

Для сохранения высокой производительности и предотвращения преждевременного утомления следует соблюдать рациональный режим работы. Непрерывная работа за компьютером не должна превышать 4 часов в первой половине дня и 3 часов - во второй. После каждого часа работы необходимо делать 5-10-минутный перерыв для отдыха. В середине рабочего дня предусматривается перерыв на обед продолжительностью от 30 до 60 минут.

При появлении признаков переутомления – головной боли, рези в глазах, дискомфорта в спине – следует немедленно сделать перерыв и выполнить комплекс упражнений для расслабления соответствующих групп мышц. Регулярное выполнение этих рекомендаций позволит минимизировать вредное воздействие факторов рабочей среды и сохранить здоровье на долгие годы.

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

Заключение

В ходе выполнения данной выпускной квалификационной работы была спроектирована и разработана программа для расчёта вероятностно-временных характеристик многопроцессорных систем на основе стохастических сетей массового обслуживания. В результате был проведён глубокий анализ многопроцессорных вычислительных систем как объектов массового обслуживания, включая исследование потоков заявок к процессорам, памяти и контроллерам ввода-вывода. Были выявлены основные недостатки существующих программных аналогов: отсутствие специализации на многопроцессорных архитектурах, сложность визуального моделирования и ограниченные возможности для исследования вероятностно-временных характеристик. Перед началом разработки был проведён анализ предметной области, выполнен обзор существующих решений и выбраны необходимые программные и аппаратные средства для создания собственной системы.

В итоге было принято решение разрабатывать приложение в операционной системе ALT Linux; вычислительная логика реализована на языке программирования C как наиболее эффективном для решения задач с большим объёмом вычислений. Графический интерфейс пользователя выполнен с использованием библиотеки GTK+ 3, что обеспечивает визуальное построение графа сети без использования внешних инструментов. Для отрисовки графа и диаграмм была выбрана библиотека Cairo. В качестве целевой платформы была выбрана ОС ALT Linux, обеспечивающая совместимость с компьютерным парком учебного заведения.

На основе приведённых выше технологий удалось спроектировать, разработать и протестировать приложение, включающее все необходимые функции для исследования многопроцессорных систем. В программе реализованы добавление, удаление и перемещение узлов на рабочей области, создание связей между узлами с заданием вероятностей переходов, редактирование параметров узлов (наименование, время обслуживания, число

					ПГУ 1.090501.12.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

каналов, тип очереди и её размер). Вычислительный модуль реализует итерационный метод расчёта интенсивностей потоков и аналитические формулы для многоканальных систем массового обслуживания с ограниченной и неограниченной очередью ($M/M/m$ и $M/M/m/K$). Результаты расчёта сохраняются в текстовый файл в виде таблицы, содержащей характеристики каждого узла (коэффициент загрузки, среднюю длину очереди, среднее время ожидания и пребывания). Реализована возможность сохранения и загрузки проектов в формате JSON, что позволяет восстанавливать ранее созданные модели. Визуализация результатов включает отображение графа сети с указанием весов связей и построение столбчатых диаграмм характеристик.

Разработанная в данной выпускной квалификационной работе программа содержит весь необходимый базовый функционал для исследования многопроцессорных систем методом стохастических сетей массового обслуживания, однако может быть дополнена. Перспективными направлениями развития являются реализация режима варьирования параметров для серий расчётов, построение графиков зависимости характеристик от параметров, расширение поддержки различных дисциплин обслуживания (приоритеты, относительные приоритеты), а также разработка веб-интерфейса для доступа к программе удалённо. Созданная программа может использоваться в учебных целях при изучении теории массового обслуживания, а также инженерами-исследователями для анализа производительности многопроцессорных систем на этапе проектирования.

Список использованных источников

1. Т. В. Черушева, Н. В. Зверовщикова, Теория массового обслуживания – Пенза Издательство ПГУ, 2021. – 224 с.
2. Архитектура ЭВМ [Электронный ресурс]. URL: <http://arch.altspu.ru/files/lection6.pdf> (дата обращения: 15.02.2026).
3. С. А. Олейникова, Математическое моделирование и системы массового обслуживания – Воронеж, 2021. – 90 с.
4. Керниган Б.У., Ритчи Д.М. Язык программирования С. –Вильямс, 2019. – 304 с.
5. Logan A. The GTK+ 3 Tutorial. – Open Source Community, 2018. – 320 с.
6. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. – IETF RFC 8259, 2017.
7. Вентцель Е.С. Исследование операций. – Советское радио, 1972. – 552 с.
8. Гнеденко Б.В., Коваленко И.Н. Введение в теорию массового обслуживания. – Наука, 1987. – 336 с.
9. Клейнрок Л. Вычислительные системы с очередями. – Мир, 1979. – 600 с.
10. Саати Т.Л. Элементы теории массового обслуживания и её приложения. – Советское радио, 1971. – 520 с.
11. Джексон Дж. Р. Сети массового обслуживания / Теория массового обслуживания. – Мир, 1966. – 114 с.
12. Башарин Г.П., Бочаров П.П., Коган Я.А. Анализ очередей в вычислительных сетях. – Наука, 1989. – 336 с.
13. Лифшиц А.Л., Мальцев Е.А. Статистическое моделирование систем массового обслуживания. – Советское радио, 1978. – 248 с.
14. Шнепс М.А. Системы распределения ресурсов и сети массового обслуживания. – Радио и связь, 1985. – 256 с.
15. Прата С. Язык программирования С. Лекции и упражнения. – М. Вильямс, 2017. – 960 с.

16. Харбисон С., Стил Г. Язык С. Справочное руководство. – Вильямс, 2014. – 448 с.
17. Подбельский В.В. Язык Си. – М.: Финансы и статистика, 2003. – 600 с.
18. Крупник А.Б. Программирование на языке С для начинающих. – Диалог-МИФИ, 2015. – 356 с.
19. Krause A. Foundations of GTK+ Development. – Apress, 2007. – 640 с.
20. Cairo Graphics Library [Электронный ресурс].
URL: <https://www.cairographics.org/documentation/> (дата обращения: 17.04.2026).
21. GTK+ Reference Manual [Электронный ресурс].
URL: <https://developer.gnome.org/gtk3/stable/> (дата обращения: 17.04.2026).
22. Шипинов О.В. Введение в GTK+. – ИНТУИТ, 2010. – 204 с.
23. Керниган Б.У., Пайк Р. UNIX. Программное окружение. – Символ-Плюс, 2008. – 416 с.
26. Немет Э., Снайдер Г., Хейн Т. Руководство администратора Linux. — Вильямс, 2018. – 1152 с.
27. JSON-Glib Reference Manual [Электронный ресурс].
URL: <https://developer.gnome.org/json-glib/stable/> (дата обращения: 17.04.2026).

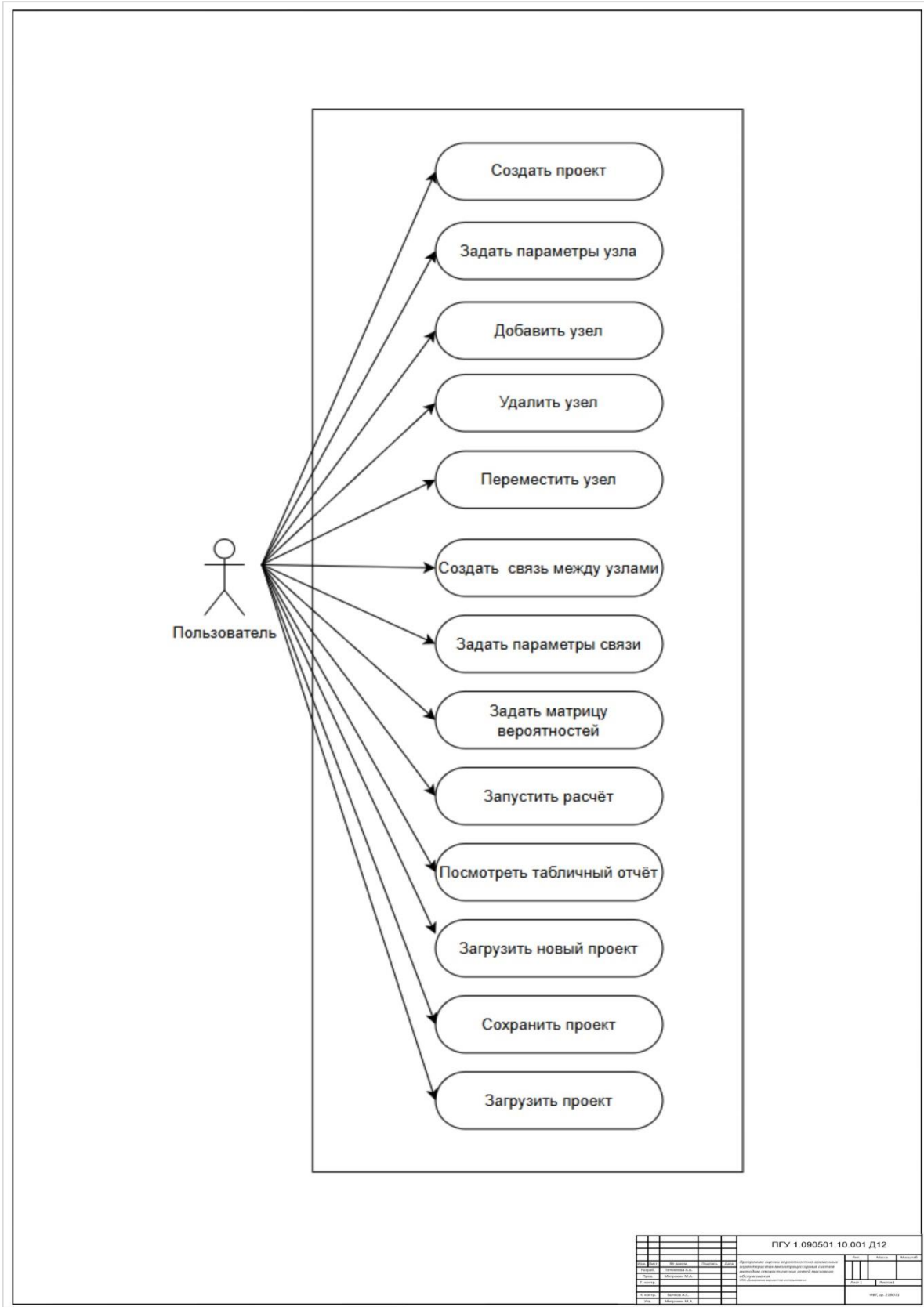
Приложение А

(обязательное)

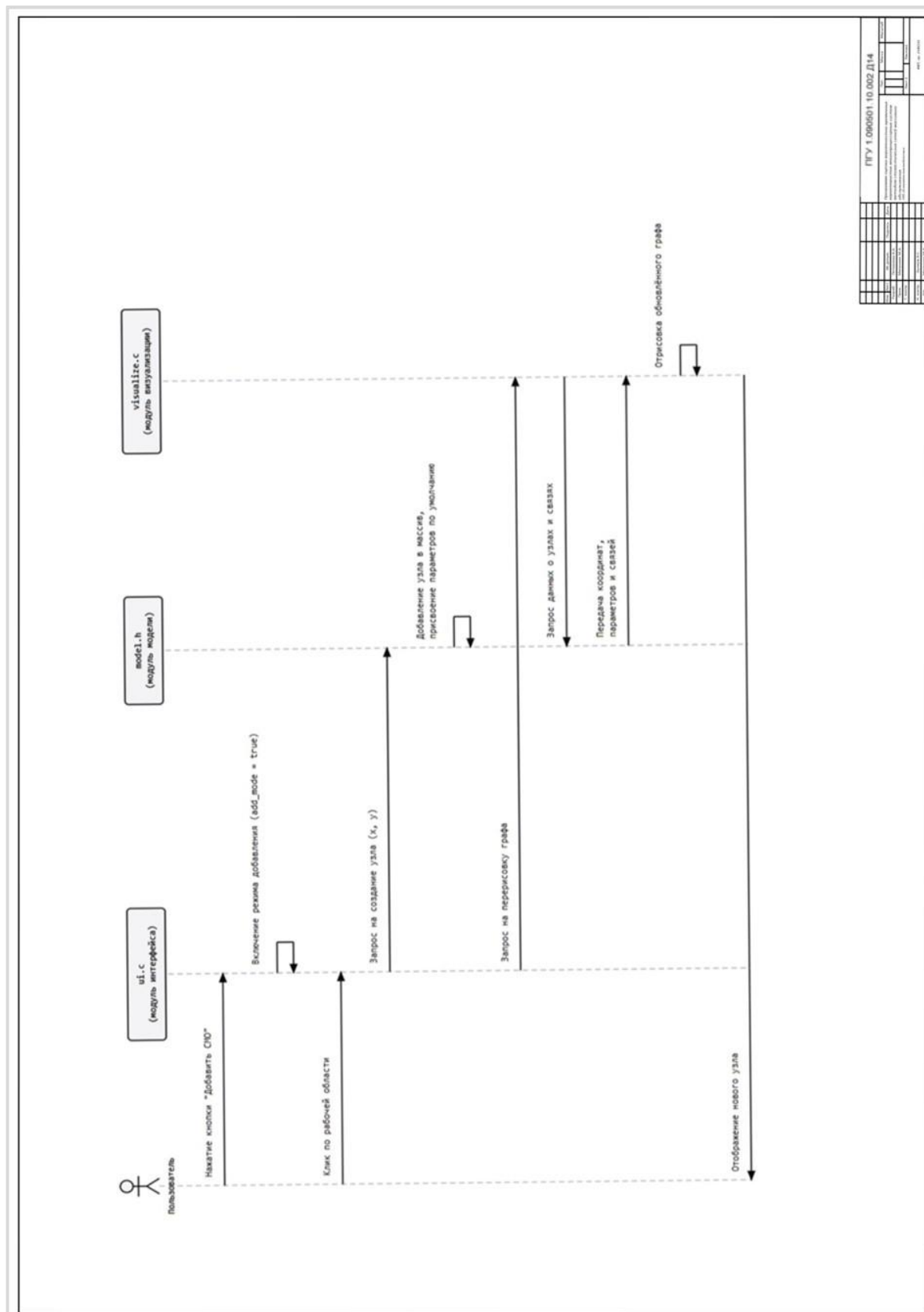
Программа оценки вероятностно-временных характеристик
многопроцессорных систем методом стохастических сетей массового
обслуживания
UML-диаграммы

					ПГУ 1.090501.12.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

UML – диаграмма вариантов использования

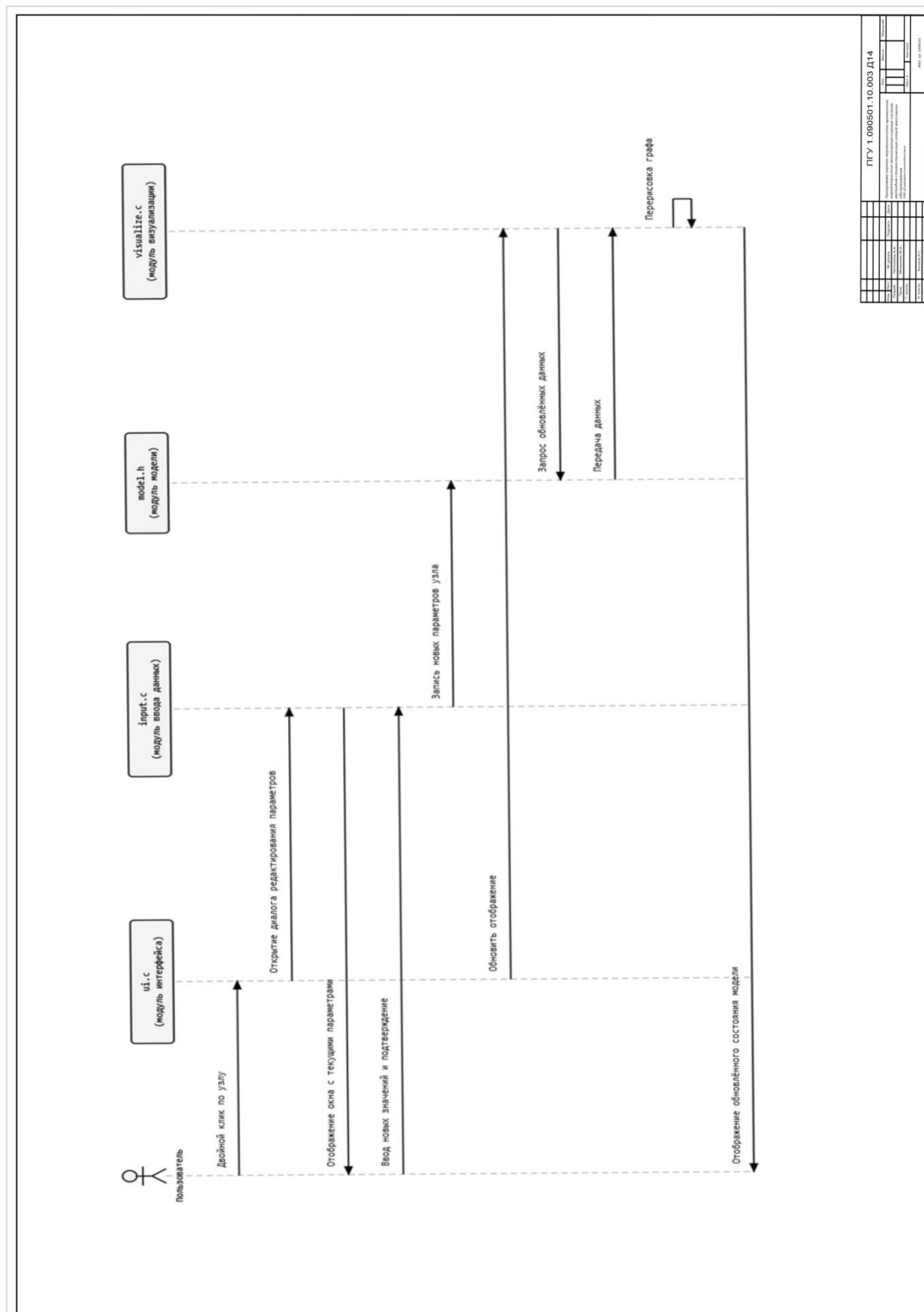


UML – диаграмма последовательности

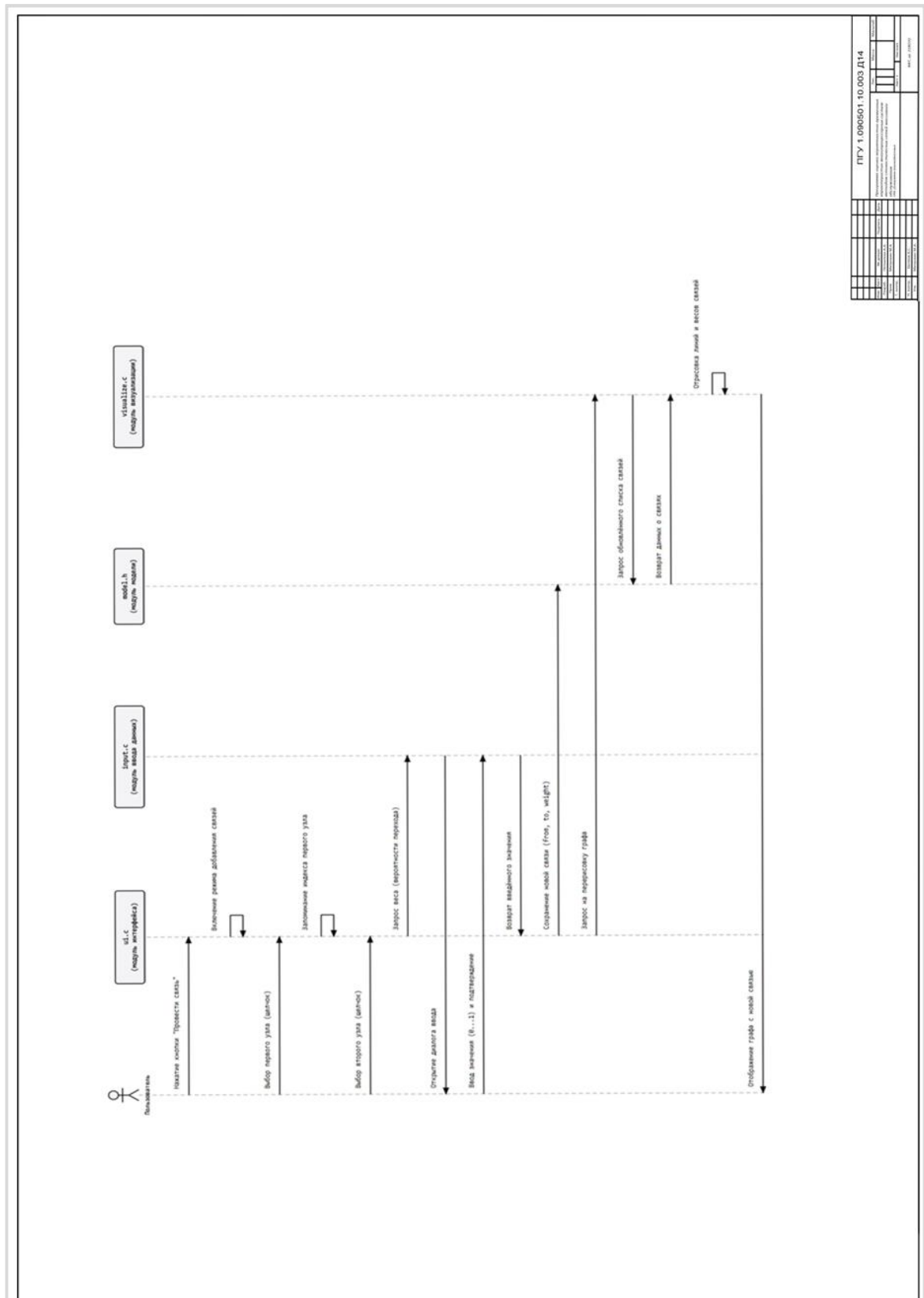


Изм.	Лист	№ докум.	Подпись	Дата

UML – диаграмма последовательности

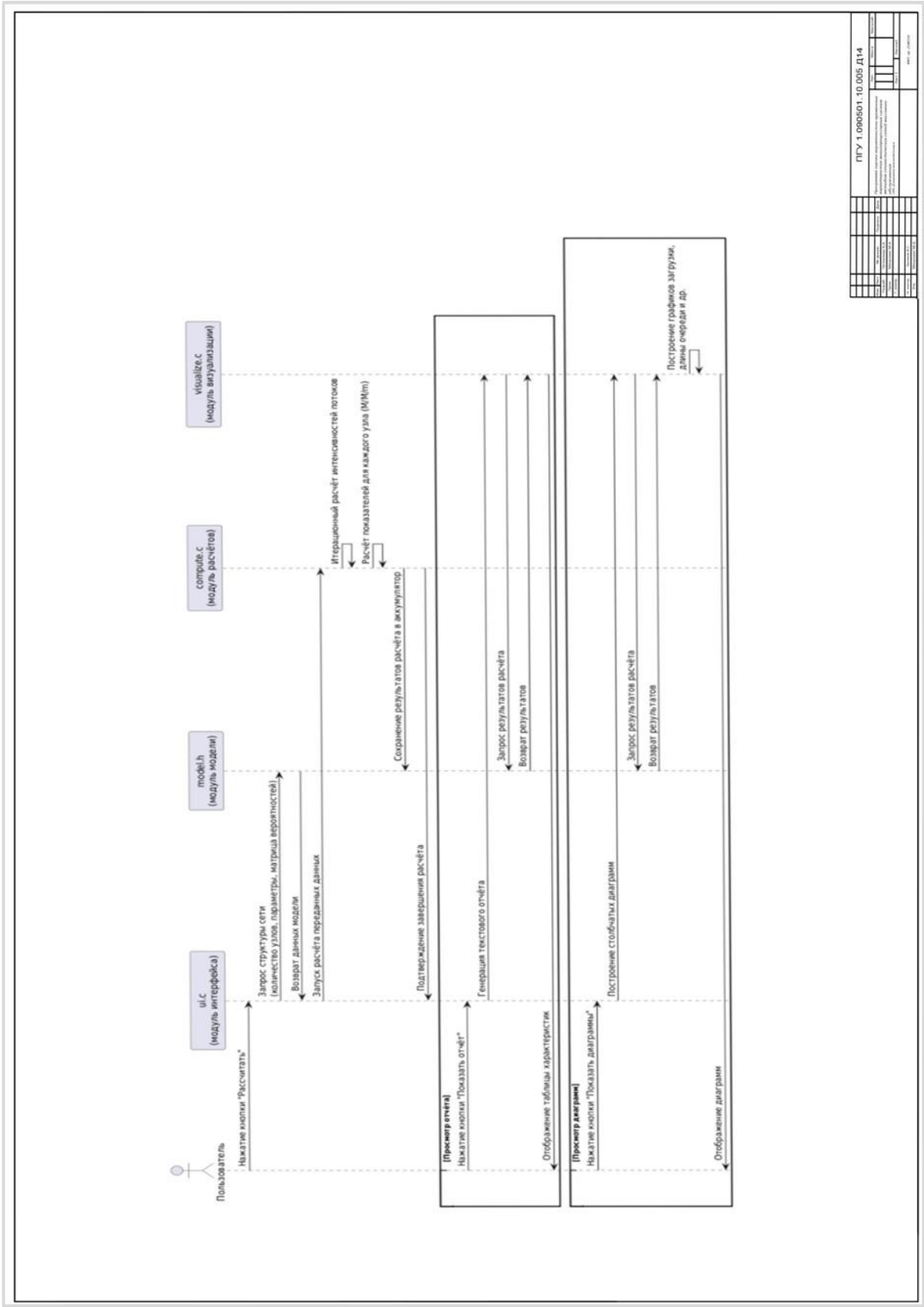


UML – диаграмма последовательности



ПГУ 1.090501.10.003 Д14	
Исполнитель	Исполнитель
Проверенный	Проверенный
Утвержденный	Утвержденный
Дата	Дата
Лист	Лист

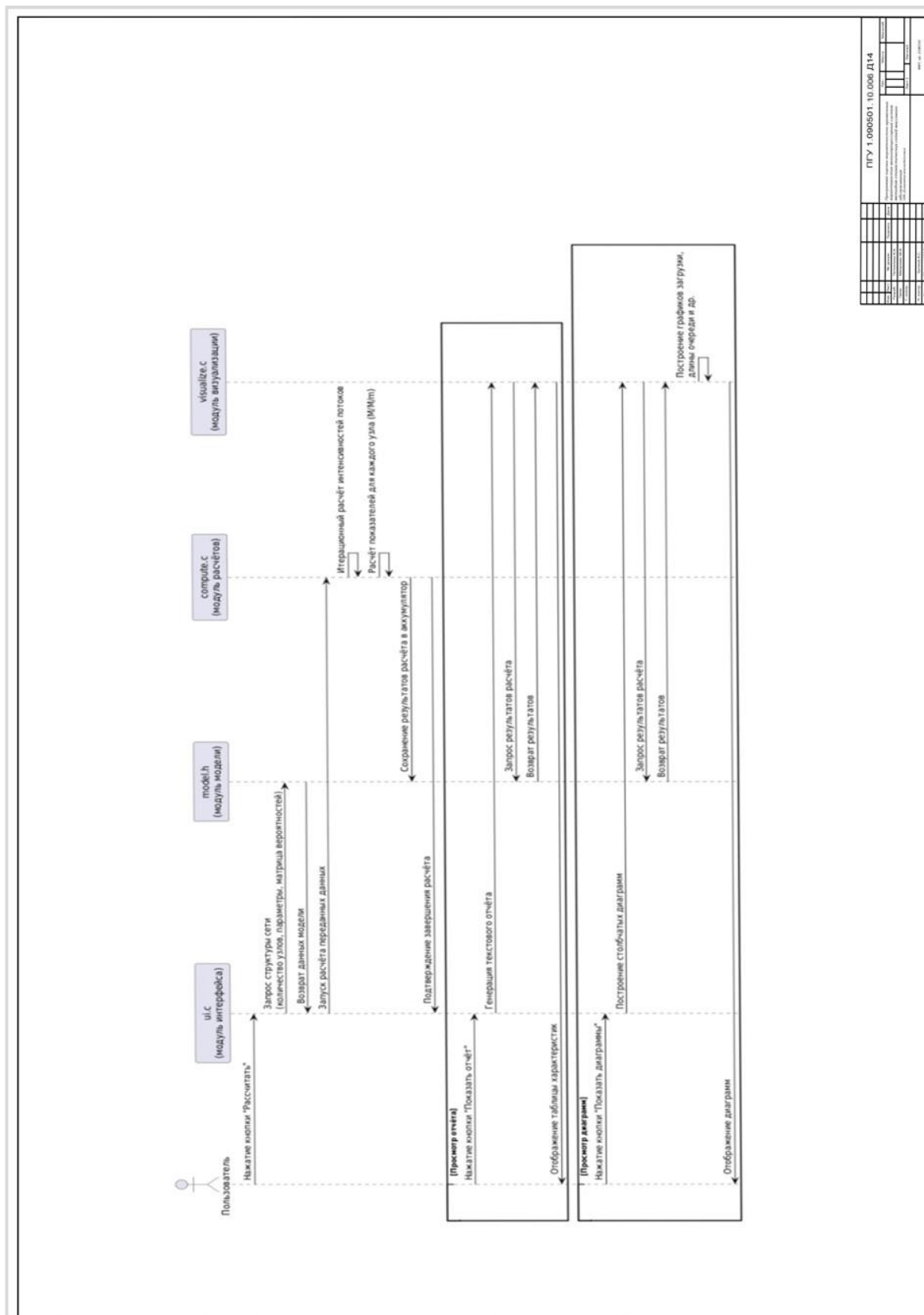
UML – диаграмма последовательности



ПУ 1.090501.10.005 Д14	
Исполнитель	Исполнитель
Проверенный	Проверенный
Утвержденный	Утвержденный
Дата	Дата

Изм.	Лист	№ докум.	Подпись	Дата

UML – диаграмма последовательности



Приложение Б

(обязательное)

Программа оценки вероятностно-временных характеристик
многопроцессорных систем методом стохастических сетей массового
обслуживания

Листинг программы

					ПГУ 1.090501.12.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «compute.c»

```
#include "compute.h"

#include "io.h"

#include <stdio.h>

#include <math.h>

// Вычисление факториала

double factorial(int n) {

    static double fact_cache[101] = {0};

    if (n < 0) return 1.0;

    if (n == 0 || n == 1) return 1.0;

    if (fact_cache[n] != 0) return fact_cache[n];

    fact_cache[n] = n * factorial(n - 1);

    return fact_cache[n];

}

// Расчёт характеристик одного узла М/М/м

CircleCalcResult calculate_mm_m(double lambda, double mu, int
m, bool limited_queue, int queue_size) {

    CircleCalcResult res = {0};

    // Проверка корректности входных данных

    if (m <= 0 || mu <= 0) {

        return res;

    }

    // Если интенсивность потока равна нулю, возвращаем
    нулевые характеристики
```

					ПГУ 1.090501.12.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

```

if (lambda <= 1e-9) {

    return res;

}

double rho = lambda / (m * mu);

res.input_intensity = lambda;

res.load_factor = rho;

res.throughput = 1.0;

// Проверка стабильности для неограниченной очереди

if (!limited_queue && rho >= 1.0 - 1e-9) {

    return res;

}

if (limited_queue) {

    //

    // М/М/м/К с ограниченной очередью

    int K = m + queue_size;

    if (K < m) K = m;

    double rho_lim = rho;

    if (rho_lim > 0.95) rho_lim = 0.95;

    double sum1 = 0.0;

    for (int n = 0; n < m; n++) {

        sum1 += pow(m * rho_lim, n) / factorial(n);

    }

    double sum2 = 0.0;

    if (fabs(rho_lim - 1.0) < 1e-9) {

```

```

        sum2 = (pow(m * rho_lim, m) / factorial(m)) * (K -
m + 1);

    } else {

        sum2 = (pow(m * rho_lim, m) / factorial(m)) * (1 -
pow(rho_lim, K - m + 1)) / (1 - rho_lim);

    }

    double P0 = 1.0 / (sum1 + sum2);

    double PK = (pow(m * rho_lim, K) / (factorial(m) *
pow(m, K - m))) * P0;

    double Ls = 0.0;

    for (int n = 0; n <= K; n++) {

        double Pn;

        if (n < m) {

            Pn = (pow(m * rho_lim, n) / factorial(n)) *
P0;

        } else {

            Pn = (pow(m, m) * pow(rho_lim, n) /
factorial(m)) * P0;

        }

        Ls += n * Pn;

    }

    double sumP0 = 0.0;

    for (int n = 0; n < m; n++) {

        sumP0 += pow(m * rho_lim, n) / factorial(n);

    }

```

```

double busy_channels = m * (1 - P0 * sumP0);

double effective_lambda = lambda * (1 - PK);

double Lq = Ls - busy_channels;

if (Lq < 0) Lq = 0;

if (Lq > 1e6) Lq = 1e4;

double W = (effective_lambda > 1e-9) ? Ls /
effective_lambda : 0;

double Wq = (effective_lambda > 1e-9) ? Lq /
effective_lambda : 0;

res.busy_channels = busy_channels;

res.avg_in_system = Ls;

res.avg_in_queue = Lq;

res.avg_wait_queue = Wq;

res.avg_time_system = W;

} else {

// М/М/м с НЕОГРАНИЧЕННОЙ очередью

double rho_lim = rho;

if (rho_lim > 0.99) rho_lim = 0.99;

double sum = 0.0;

for (int r = 0; r < m; r++) {

    sum += pow(m * rho_lim, r) / factorial(r);

}

double pm = pow(m * rho_lim, m) / (factorial(m) * (1 -
rho_lim));

double P0 = 1.0 / (sum + pm);

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

```

        // Формула из документа:  $l_i = (m * \rho^{(m+1)} / (m! * (1-\rho)^{(m+1)})) * \pi_0$ 
        double numerator = m * pow(rho_lim, m + 1);

        double denominator = factorial(m) * pow(1 - rho_lim, m
+ 1);

        double Lq = (numerator / denominator) * P0;

        if (isnan(Lq) || isinf(Lq)) {

            Lq = 0.0;

        }

        if (Lq > 1e6) {

            Lq = 1e4;

        }

        double Wq = (lambda > 1e-9) ? Lq / lambda : 0.0;

        double W = Wq + 1.0 / mu;

        double busy_channels = m * rho;

        double L = Lq + busy_channels;

        res.busy_channels = busy_channels;

        res.avg_in_system = L;

        res.avg_in_queue = Lq;

        res.avg_wait_queue = Wq;

        res.avg_time_system = W;

    }

    return res;

}

```

					ПГУ 1.090501.12.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		101

```

// Расчёт всей сети (исправленная версия)

void calculate_network(Circles* circles, double
current_intensity, int cycle_index) {

    if (!circles) return;

    if (circles->count < 1) return;

    double lambda[MAX_CIRCLES] = {0};

    const int max_iter = 1000;           // Увеличили до 1000

    const double epsilon = 1e-10;       // Более строгий
критерий

    // Проверка суммы весов исходящих связей

    for (int i = 0; i < circles->count; i++) {

        double sum_weights = 0.0;

        for (int j = 0; j < circles->line_count; j++) {

            if (circles->lines[j].from_index == i) {

                sum_weights += circles->lines[j].weight;

            }

        }

        if (sum_weights > 0 && fabs(sum_weights - 1.0) > 1e-5)
{

            printf("Предупреждение: сумма весов для узла S%d =
%.3f (должна быть 1.0)\n",

                i, sum_weights);

        }

    }

    // Начальное приближение для интенсивностей

```

```

    lambda[0] = current_intensity;

    // Важно: задаём начальное приближение для всех узлов

    // Это ускоряет сходимость и помогает найти правильное
решение

    for (int i = 1; i < circles->count; i++) {

        lambda[i] = current_intensity;

    }

    printf("\n=== ИТЕРАЦИОННЫЙ РАСЧЁТ ИНТЕНСИВНОСТЕЙ ===\n");

    printf("Входящая интенсивность  $\Lambda$  = %.4f\n",
current_intensity);

    printf("Количество узлов: %d, связей: %d\n", circles-
>count, circles->line_count);

    // Итерационный расчёт без демпфирования

    for (int iter = 0; iter < max_iter; iter++) {

        double new_lambda[MAX_CIRCLES] = {0};

        new_lambda[0] = current_intensity;

        // Вычисляем новые значения интенсивностей

        for (int i = 0; i < circles->line_count; i++) {

            int from = circles->lines[i].from_index;

            int to = circles->lines[i].to_index;

            double weight = circles->lines[i].weight;

            if (weight > 0) {

                new_lambda[to] += lambda[from] * weight;

            }

        }

    }

```

```

// Проверяем изменение

double max_diff = 0.0;

for (int i = 1; i < circles->count; i++) {

    double diff = fabs(new_lambda[i] - lambda[i]);

    if (diff > max_diff) max_diff = diff;

    lambda[i] = new_lambda[i];

}

if (iter < 5 || iter % 50 == 0) {

    printf("Итерация %3d: max_diff = %.2e\n", iter +
1, max_diff);

}

if (max_diff < epsilon) {

    printf("\n*** СХОДИМОСТЬ ДОСТИГНУТА за %d итераций
***\n", iter + 1);

    break;

}

if (iter == max_iter - 1) {

    printf("\n!!! ПРЕДУПРЕЖДЕНИЕ: Сходимость не
достигнута за %d итераций !!!\n", max_iter);

    printf("Последнее max_diff = %.2e\n", max_diff);

}

}

// Вывод рассчитанных интенсивностей

printf("\n=== РАССЧИТАННЫЕ ИНТЕНСИВНОСТИ ===\n");

for (int i = 0; i < circles->count; i++) {

```

```

        printf("S%d:  $\lambda$  = %.4f\n", i, lambda[i]);
    }

    // Расчёт характеристик для каждого узла
    for (int i = 0; i < circles->count; i++) {

        CircleParams* p = &circles->circle_params[i];

        double mu = (p->service_time > 0) ? 1.0 / p-
>service_time : 1.0;

        int m = (p->channel_count > 0) ? p->channel_count : 1;

        CircleCalcResult res = calculate_mm_m(lambda[i], mu,
m,

p-
>limited_queue, p->queue_size);

        if (current_intensity > 0) {

            res.throughput = lambda[i] / current_intensity;

        } else {

            res.throughput = 0;

        }

        res.input_intensity = lambda[i];

        results_acc.results[i][cycle_index] = res;

    }

    // Сохраняем результаты в файл
    if (cycle_index == 0 || cycles_count == 1) {

        save_results_to_file("result.txt", circles);

    }

```

```

}

// Расчёт с варьированием параметров

void vary_and_calculate(Circles* circles) {

    if (!circles) return;

    if (cycles_count < 1) cycles_count = 1;

    if (cycles_count > MAX_CYCLES) cycles_count = MAX_CYCLES;

    results_acc.cycle_count = cycles_count;

    results_acc.circle_count = circles->count;

    printf("\n=== НАЧАЛО РАСЧЁТА С ВАРЬИРОВАНИЕМ ПАРАМЕТРОВ
===\n");

    printf("Количество циклов: %d\n\n", cycles_count);

    for (int cycle = 0; cycle < cycles_count; cycle++) {

        double t = (cycles_count == 1) ? 0.0 : (double)cycle /
(cycles_count - 1);

        double current_intensity = input_intensity;

        if (vary_intensity) {

            current_intensity = intensity_from + t *
(intensity_to - intensity_from);

        }

        for (int i = 0; i < circles->count; i++) {

            CircleParams* p = &circles->circle_params[i];

            if (vary_service_time) {

                p->service_time = service_time_from[i] + t *
(service_time_to[i] - service_time_from[i]);

```

```

        if (p->service_time < 0.0001) p->service_time
= 0.0001;

    }

    if (vary_channel_count) {

        float val = channel_count_from[i] + t *
(channel_count_to[i] - channel_count_from[i]);

        int m = (int)(val + 0.5);

        if (m < 1) m = 1;

        p->channel_count = m;

    }

    if (vary_queue_size) {

        float val = queue_size_from[i] + t *
(queue_size_to[i] - queue_size_from[i]);

        int m = (int)(val + 0.5);

        if (m < 1) m = 1;

        p->queue_size = m;

    }

}

printf("=== Цикл %d / %d ===\n", cycle + 1,
cycles_count);

printf("Параметры цикла:\n");

printf("    Интенсивность входного потока: %.4f\n",
current_intensity);

for (int i = 0; i < circles->count; i++) {

    CircleParams* p = &circles->circle_params[i];

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						107
Изм.	Лист	№ докум.	Подпись	Дата		

```

        printf("  S%d: время_обсл=%.4f, каналов=%d,
очередь=%d\n",

                i, p->service_time, p->channel_count, p-
>queue_size);

    }

    calculate_network(circles, current_intensity, cycle);

    printf("\n");

}

printf("=== СРЕДНИЕ ЗНАЧЕНИЯ ПО %d ЦИКЛАМ ===\n",
cycles_count);

for (int i = 0; i < circles->count; i++) {

    CircleCalcResult avg = {0};

    for (int cycle = 0; cycle < cycles_count; cycle++) {

        avg.throughput +=
results_acc.results[i][cycle].throughput;

        avg.input_intensity +=
results_acc.results[i][cycle].input_intensity;

        avg.load_factor +=
results_acc.results[i][cycle].load_factor;

        avg.busy_channels +=
results_acc.results[i][cycle].busy_channels;

        avg.avg_in_system +=
results_acc.results[i][cycle].avg_in_system;

        avg.avg_in_queue +=
results_acc.results[i][cycle].avg_in_queue;

        avg.avg_wait_queue +=
results_acc.results[i][cycle].avg_wait_queue;

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						108
Изм.	Лист	№ докум.	Подпись	Дата		

```

        avg.avg_time_system +=
results_acc.results[i][cycle].avg_time_system;

    }

    double c = (double)cycles_count;

    printf("Узел S%d (среднее):\n", i);

    printf("  α: %.4f\n", avg.throughput / c);

    printf("  λ: %.4f\n", avg.input_intensity / c);

    printf("  ρ: %.4f\n", avg.load_factor / c);

    printf("  k: %.4f\n", avg.busy_channels / c);

    printf("  r: %.4f\n", avg.avg_in_system / c);

    printf("  l: %.4f\n", avg.avg_in_queue / c);

    printf("  ω: %.4f\n", avg.avg_wait_queue / c);

    printf("  u: %.4f\n\n", avg.avg_time_system / c);

}

}

```

Файл «main.c»

```

#include "model.h"

#include "ui.h"

#include "visualize.h"

#include "input.h"

#include "compute.h"

#include "io.h"

#include <stdio.h>

// Глобальные переменные (определение)

```

```

float input_intensity = 1.0f;

int cycles_count = 2;

bool project_modified = false;

CalcResultsAccumulator results_acc = {0};

bool vary_intensity = false;

float intensity_from = 0.0f, intensity_to = 0.0f;

bool vary_service_time = false;

float service_time_from[MAX_CIRCLES] = {0};

float service_time_to[MAX_CIRCLES] = {0};

bool vary_channel_count = false;

int channel_count_from[MAX_CIRCLES] = {0};

int channel_count_to[MAX_CIRCLES] = {0};

bool vary_queue_size = false;

int queue_size_from[MAX_CIRCLES] = {0};

int queue_size_to[MAX_CIRCLES] = {0};

GtkWidget* matrix_window = NULL;

Circles* current_circles = NULL;

const char* characteristic_names[CHAR_COUNT] = {

    "Коефф. передачи",

    "Интенсивность входного потока",

    "Коефф. загрузки",

    "Ср. число занятых каналов",

    "Ср. число заявок в СМО",

    "Ср. длина очереди",

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

```

        "Ср. время ожидания в очереди",

        "Ср. время пребывания в СМО"

};

// Функция создания главного окна

static void activate(GtkApplication* app, gpointer user_data)
{
    // Выделяем память под модель

    current_circles = (Circles*)calloc(1, sizeof(Circles));

    if (!current_circles) {

        g_printerr("Ошибка выделения памяти\n");

        return;

    }

    clear_circles(current_circles);

    // Создаём главное окно

    GtkWidget* window = gtk_application_window_new(app);

    gtk_window_set_title(GTK_WINDOW(window), "Расчёт
стохастических сетей массового обслуживания");

    gtk_window_set_default_size(GTK_WINDOW(window), 1200,
800);

    // Вертикальный контейнер

    GtkWidget* vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL,
0);

    gtk_container_add(GTK_CONTAINER(window), vbox);

    // Меню и панель инструментов

    GtkWidget* menubar = create_menubar(current_circles);

```

```

    gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE,
0);

    GtkWidget* toolbar = create_toolbar(current_circles);

    gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE,
0);

    // Область для рисования

    GtkWidget* drawing_area = gtk_drawing_area_new();

    gtk_widget_set_size_request(drawing_area, 1200, 700);

    gtk_box_pack_start(GTK_BOX(vbox), drawing_area, TRUE,
TRUE, 0);

    // Подключаем обработчики событий

    g_signal_connect(G_OBJECT(drawing_area), "draw",

                    G_CALLBACK(on_draw_event),
current_circles);

    g_signal_connect(drawing_area, "button-press-event",

                    G_CALLBACK(on_button_press),
current_circles);

    g_signal_connect(drawing_area, "motion-notify-event",

                    G_CALLBACK(on_motion_notify),
current_circles);

    g_signal_connect(drawing_area, "button-release-event",

                    G_CALLBACK(on_button_release),
current_circles);

    gtk_widget_add_events(drawing_area, GDK_BUTTON_PRESS_MASK
|

    GDK_BUTTON_RELEASE_MASK | GDK_POINTER_MOTION_MASK);

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						112
Изм.	Лист	№ докум.	Подпись	Дата		

```

        g_signal_connect(window, "destroy",
G_CALLBACK(gtk_main_quit), NULL);

        gtk_widget_show_all(window);

    }

    int main(int argc, char** argv) {

        GtkApplication* app;

        int status;

        app = gtk_application_new("org.altlinux.smo",
G_APPLICATION_FLAGS_NONE);

        g_signal_connect(app, "activate", G_CALLBACK(activate),
NULL);

        status = g_application_run(G_APPLICATION(app), argc,
argv);

        g_object_unref(app);

        // Освобождаем память при выходе

        if (current_circles) {

            clear_circles(current_circles);

            free(current_circles);

            current_circles = NULL;

        }

        return status;

    }

```

Файл «model.c »

```

#include "model.h"

#include <stdlib.h>

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						113
Изм.	Лист	№ докум.	Подпись	Дата		

```

#include <string.h>

void clear_circles(Circles* circles) {

    if (!circles) return;

    // Освобождаем память имён узлов

    for (int i = 0; i < circles->count; i++) {

        if (circles->circle_params[i].circle_name) {

            free(circles->circle_params[i].circle_name);

            circles->circle_params[i].circle_name = NULL;

        }

        if (circles->circle_params[i].priority_table) {

            free(circles->circle_params[i].priority_table);

            circles->circle_params[i].priority_table = NULL;

        }

        circles->x[i] = 0;

        circles->y[i] = 0;

    }

    circles->count = 0;

    circles->line_count = 0;

    circles->add_mode = false;

    circles->del_mode = false;

    circles->move_mode = false;

    circles->add_line_mode = false;

    circles->delLine_mode = false;

    circles->line_first_circle = -1;

```

```

        circles->moving_index = -1;

        // Обнуляем массивы

        memset(circles->x, 0, sizeof(circles->x));

        memset(circles->y, 0, sizeof(circles->y));

        memset(circles->lines, 0, sizeof(circles->lines));

    }

    void delete_circle_and_lines(Circles* circles, int
circle_index) {

        if (!circles || circle_index < 0 || circle_index >=
circles->count) return;

        // Удаляем связи, связанные с этим узлом

        for (int i = 0; i < circles->line_count; ) {

            if (circles->lines[i].from_index == circle_index ||

                circles->lines[i].to_index == circle_index) {

                for (int j = i; j < circles->line_count - 1; j++)

{

                    circles->lines[j] = circles->lines[j + 1];

                }

                circles->line_count--;

            } else {

                i++;

            }

        }

        // Освобождаем имя узла

        if (circles->circle_params[circle_index].circle_name) {

```

```

        free(circles->circle_params[circle_index].circle_name);
    }

    // Сдвигаем узлы
    for (int i = circle_index; i < circles->count - 1; i++) {
        circles->x[i] = circles->x[i + 1];
        circles->y[i] = circles->y[i + 1];
        circles->circle_params[i] = circles->circle_params[i +
1];
    }

    circles->count--;

    // Корректируем индексы в связях
    for (int i = 0; i < circles->line_count; i++) {
        if (circles->lines[i].from_index > circle_index) {
            circles->lines[i].from_index--;
        }

        if (circles->lines[i].to_index > circle_index) {
            circles->lines[i].to_index--;
        }
    }
}

void free_priority_table(CircleParams* params) {
    if (params->priority_table) {
        free(params->priority_table);
    }
}

```

```

        params->priority_table = NULL;

    }

}

```

Файл « ui.c »

```

#include "ui.h"

#include "input.h"

#include "compute.h"

#include "io.h"

#include "visualize.h"

#include <stdio.h>

#include <stdlib.h>

// Структура для передачи данных в обработчик сохранения
матрицы

typedef struct {

    Circles* circles;

    GtkWidget*** spin_buttons;

    int node_count;

} MatrixData;

// Глобальная переменная для окна матрицы

extern GtkWidget* matrix_window;

// Включение/выключение режимов редактирования

void enable_add_mode(Circles* circles) {

    if (!circles) return;

    circles->add_mode = true;

```

```

        circles->del_mode = false;

        circles->move_mode = false;

        circles->add_line_mode = false;

        circles->delLine_mode = false;

        printf("Режим добавления включен\n");
    }

void enable_del_mode(Circles* circles) {

    if (!circles) return;

    circles->del_mode = true;

    circles->add_mode = false;

    circles->move_mode = false;

    circles->add_line_mode = false;

    circles->delLine_mode = false;

    printf("Режим удаления включен\n");

}

void enable_move_mode(Circles* circles) {

    if (!circles) return;

    circles->move_mode = true;

    circles->add_mode = false;

    circles->del_mode = false;

    circles->add_line_mode = false;

    circles->delLine_mode = false;

    circles->moving_index = -1;

    printf("Режим перемещения включен\n");
}

```

```

}

void enable_add_line_mode(Circles* circles) {

    if (!circles) return;

    circles->add_line_mode = true;

    circles->line_first_circle = -1;

    circles->add_mode = false;

    circles->del_mode = false;

    circles->move_mode = false;

    circles->delLine_mode = false;

    printf("Режим добавления связи включен\n");

}

void enable_del_line_mode(Circles* circles) {

    if (!circles) return;

    circles->delLine_mode = true;

    circles->add_mode = false;

    circles->del_mode = false;

    circles->move_mode = false;

    circles->add_line_mode = false;

    printf("Режим удаления связи включен\n");

}

// Обработчики кнопок

void on_add_cmo_clicked(GtkButton* button, gpointer user_data)
{

    printf("Кнопка Добавить СМО нажата\n");

```

```

        Circles* circles = (Circles*)user_data;

        enable_add_mode(circles);

    }

void on_del_cmo_clicked(GtkButton* button, gpointer user_data)
{

    printf("Кнопка Удалить СМО нажата\n");

    Circles* circles = (Circles*)user_data;

    enable_del_mode(circles);

}

void on_move_cmo_clicked(GtkButton* button, gpointer
user_data) {

    printf("Кнопка Переместить СМО нажата\n");

    Circles* circles = (Circles*)user_data;

    enable_move_mode(circles);

}

void on_add_line_clicked(GtkButton* button, gpointer
user_data) {

    printf("Кнопка Провести связь нажата\n");

    Circles* circles = (Circles*)user_data;

    enable_add_line_mode(circles);

}

void on_del_line_clicked(GtkButton* button, gpointer
user_data) {

    printf("Кнопка Удалить связь нажата\n");

    Circles* circles = (Circles*)user_data;

```

```

        enable_del_line_mode(circles);

    }

    void on_calc_net_clicked(GtkButton* button, gpointer
user_data) {

        printf("Кнопка Рассчитать нажата\n");

        Circles* circles = (Circles*)user_data;

        if (circles && circles->count > 0) {

            calculate_network(circles, input_intensity, 0);

            save_results_to_file("result.txt", circles);

        } else {

            printf("Нет узлов для расчёта\n");

        }

    }

    void on_save_project_clicked(GtkButton* button, gpointer
user_data) {

        printf("Сохранение проекта\n");

        Circles* circles = (Circles*)user_data;

        GtkWidget* dialog = gtk_file_chooser_dialog_new(

            "Сохранить проект",

GTK_WINDOW(gtk_widget_get_toplevel(GTK_WIDGET(button))),

GTK_FILE_CHOOSER_ACTION_SAVE,

                                                                    "_Отмена",

GTK_RESPONSE_CANCEL,

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						121
Изм.	Лист	№ докум.	Подпись	Дата		

```

"_Сохранить", GTK_RESPONSE_ACCEPT,

NULL);

GtkFileFilter* filter = gtk_file_filter_new();

gtk_file_filter_set_name(filter, "JSON files");

gtk_file_filter_add_pattern(filter, "*.json");

gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog),
filter);

gtk_file_chooser_set_current_name(GTK_FILE_CHOOSER(dialog),
"project.json");

if (gtk_dialog_run(GTK_DIALOG(dialog)) ==
GTK_RESPONSE_ACCEPT) {

    char* filename =
gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));

    if (filename) {

        if (save_circles_to_file(circles, filename)) {

            printf("Проект сохранён в %s\n", filename);

            project_modified = false;

        } else {

            GtkWidget* error_dialog =
gtk_message_dialog_new(

GTK_WINDOW(gtk_widget_get_toplevel(GTK_WIDGET(button))),

GTK_DIALOG_MODAL,

GTK_MESSAGE_ERROR,

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		

```

GTK_BUTTONS_OK,

"Не удалось сохранить файл: %s", filename);

        gtk_dialog_run(GTK_DIALOG(error_dialog));

        gtk_widget_destroy(error_dialog);

    }

    g_free(filename);

}

}

gtk_widget_destroy(dialog);

}

void on_open_project_clicked(GtkButton* button, gpointer
user_data) {

    printf("Открытие проекта\n");

    Circles* circles = (Circles*)user_data;

    GtkWidget* dialog = gtk_file_chooser_dialog_new(

        "Открыть проект",

GTK_WINDOW(gtk_widget_get_toplevel(GTK_WIDGET(button))),

GTK_FILE_CHOOSER_ACTION_OPEN,

                                                                    "_Отмена",

GTK_RESPONSE_CANCEL,

"_Открыть", GTK_RESPONSE_ACCEPT,

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						123
Изм.	Лист	№ докум.	Подпись	Дата		

```

NULL);

GtkFileFilter* filter = gtk_file_filter_new();

gtk_file_filter_set_name(filter, "JSON files");

gtk_file_filter_add_pattern(filter, "*.json");

gtk_file_chooser_add_filter(GTK_FILE_CHOOSER(dialog),
filter);

if (gtk_dialog_run(GTK_DIALOG(dialog)) ==
GTK_RESPONSE_ACCEPT) {

    char* filename =
gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));

    if (filename) {

        if (load_circles_from_file(filename, circles)) {

            printf("Проект загружен из %s\n", filename);

            project_modified = false;

            GtkWidget* window =
gtk_widget_get_toplevel(GTK_WIDGET(button));

            gtk_widget_queue_draw(window);

        } else {

            GtkWidget* error_dialog =
gtk_message_dialog_new(

GTK_WINDOW(gtk_widget_get_toplevel(GTK_WIDGET(button))),

GTK_DIALOG_MODAL,

GTK_MESSAGE_ERROR,

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						124
Изм.	Лист	№ докум.	Подпись	Дата		

```

GTK_BUTTONS_OK,

"Не удалось загрузить файл: %s", filename);

        gtk_dialog_run(GTK_DIALOG(error_dialog));

        gtk_widget_destroy(error_dialog);

    }

    g_free(filename);

}

}

gtk_widget_destroy(dialog);

}

void on_new_project_clicked(GtkButton* button, gpointer
user_data) {

    printf("Новый проект\n");

    Circles* circles = (Circles*)user_data;

    clear_circles(circles);

    GtkWidget* window =
gtk_widget_get_toplevel(GTK_WIDGET(button));

    if (window) gtk_widget_queue_draw(window);

    project_modified = false;

}

// Создание панели инструментов

GtkWidget* create_toolbar(Circles* circles) {

    GtkWidget* toolbar = gtk_toolbar_new();

```

```

        gtk_toolbar_set_style(GTK_TOOLBAR(toolbar),
GTK_TOOLBAR_BOTH_HORIZ);

        // Кнопка "Добавить СМО"

        GtkToolItem* add_btn = gtk_tool_button_new(NULL, "Добавить
СМО");

        g_signal_connect(add_btn, "clicked",
G_CALLBACK(on_add_cmo_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), add_btn, -1);

        // Кнопка "Удалить СМО"

        GtkToolItem* del_btn = gtk_tool_button_new(NULL, "Удалить
СМО");

        g_signal_connect(del_btn, "clicked",
G_CALLBACK(on_del_cmo_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), del_btn, -1);

        // Кнопка "Переместить"

        GtkToolItem* move_btn = gtk_tool_button_new(NULL,
"Переместить");

        g_signal_connect(move_btn, "clicked",
G_CALLBACK(on_move_cmo_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), move_btn, -1);

        // Разделитель

        GtkToolItem* sep = gtk_separator_tool_item_new();

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);

        // Кнопка "Провести связь"

        GtkToolItem* link_btn = gtk_tool_button_new(NULL,
"Провести связь");

```

```

        g_signal_connect(link_btn, "clicked",
G_CALLBACK(on_add_line_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), link_btn, -1);

        // Кнопка "Удалить связь"

        GtkToolItem* unlink_btn = gtk_tool_button_new(NULL,
"Удалить связь");

        g_signal_connect(unlink_btn, "clicked",
G_CALLBACK(on_del_line_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), unlink_btn, -1);

        // ===== КНОПКА "МАТРИЦА СВЯЗЕЙ" (ПОСЛЕ УДАЛЕНИЯ
СВЯЗИ) =====

        GtkToolItem* matrix_btn = gtk_tool_button_new(NULL,
"Матрица связей");

        g_signal_connect(matrix_btn, "clicked",
G_CALLBACK(open_edit_weights_window), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), matrix_btn, -1);

        // Разделитель

        sep = gtk_separator_tool_item_new();

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);

        // Кнопка "Рассчитать"

        GtkToolItem* calc_btn = gtk_tool_button_new(NULL,
"Рассчитать");

        g_signal_connect(calc_btn, "clicked",
G_CALLBACK(on_calc_net_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), calc_btn, -1);

        // Кнопка "Сохранить"

```

```

        GtkToolItem* save_btn = gtk_tool_button_new(NULL,
"Сохранить");

        g_signal_connect(save_btn, "clicked",
G_CALLBACK(on_save_project_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), save_btn, -1);

        // Кнопка "Открыть"

        GtkToolItem* open_btn = gtk_tool_button_new(NULL,
"Открыть");

        g_signal_connect(open_btn, "clicked",
G_CALLBACK(on_open_project_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), open_btn, -1);

        // Кнопка "Новый"

        GtkToolItem* new_btn = gtk_tool_button_new(NULL, "Новый");

        g_signal_connect(new_btn, "clicked",
G_CALLBACK(on_new_project_clicked), circles);

        gtk_toolbar_insert(GTK_TOOLBAR(toolbar), new_btn, -1);

        return toolbar;

    }

    // Создание строки меню (упрощённое, без "О программе")

    GtkWidget* create_menubar(Circles* circles) {

        GtkWidget* menubar = gtk_menu_bar_new();

        GtkWidget* file_menu = gtk_menu_new();

        GtkWidget* file_item =
gtk_menu_item_new_with_label("Файл");

        gtk_menu_item_set_submenu(GTK_MENU_ITEM(file_item),
file_menu);

```

```

        GtkWidget* exit_item =
gtk_menu_item_new_with_label("Выход");

        g_signal_connect(exit_item, "activate",
G_CALLBACK(gtk_main_quit), NULL);

        gtk_menu_shell_append(GTK_MENU_SHELL(file_menu),
exit_item);

        gtk_menu_shell_append(GTK_MENU_SHELL(menubar), file_item);

        return menubar;

    }

    // Функция открытия окна редактирования матрицы

    void open_edit_weights_window(GtkWidget* button, gpointer
user_data) {

        Circles* circles = (Circles*)user_data;

        // Если окно уже открыто, просто показываем его

        if (matrix_window != NULL) {

            gtk_window_present(GTK_WINDOW(matrix_window));

            return;

        }

        // Создаём новое окно

        matrix_window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

        gtk_window_set_title(GTK_WINDOW(matrix_window), "Редактор
матрицы вероятностей передач");

        gtk_window_set_default_size(GTK_WINDOW(matrix_window),
700, 500);

        gtk_container_set_border_width(GTK_CONTAINER(matrix_window), 10);

```

```

        // Контейнер

        GtkWidget* vbox = gtk_box_new(GTK_ORIENTATION_VERTICAL,
6);

        gtk_container_add(GTK_CONTAINER(matrix_window), vbox);

        // Область прокрутки (если узлов много)

        GtkWidget* scrolled = gtk_scrolled_window_new(NULL, NULL);

        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(scrolled),

                                         GTK_POLICY_AUTOMATIC,
GTK_POLICY_AUTOMATIC);

        gtk_box_pack_start(GTK_BOX(vbox), scrolled, TRUE, TRUE,
0);

        // Таблица (Grid)

        GtkWidget* grid = gtk_grid_new();

        gtk_grid_set_row_spacing(GTK_GRID(grid), 5);

        gtk_grid_set_column_spacing(GTK_GRID(grid), 5);

        gtk_container_add(GTK_CONTAINER(scrolled), grid);

        int node_count = circles->count;

        // Заголовки строк и столбцов

        for (int i = 0; i < node_count; i++) {

            char label_text[16];

            snprintf(label_text, sizeof(label_text), "S%d", i);

            // Заголовки столбцов (первая строка)

            GtkWidget* col_label = gtk_label_new(label_text);

```

```

        gtk_grid_attach(GTK_GRID(grid), col_label, i + 1, 0,
1, 1);

        // Заголовки строк (первый столбец)

        GtkWidget* row_label = gtk_label_new(label_text);

        gtk_grid_attach(GTK_GRID(grid), row_label, 0, i + 1,
1, 1);

    }

    // Массив для хранения spin-кнопок

    GtkWidget*** spin_buttons = g_new0(GtkWidget**,
node_count);

    for (int i = 0; i < node_count; i++) {

        spin_buttons[i] = g_new0(GtkWidget*, node_count);

    }

    // Создаём ячейки таблицы

    for (int i = 0; i < node_count; i++) {

        for (int j = 0; j < node_count; j++) {

            // Ищем существующую связь

            double weight = 0.0;

            for (int k = 0; k < circles->line_count; k++) {

                if (circles->lines[k].from_index == i &&

                    circles->lines[k].to_index == j) {

                    weight = circles->lines[k].weight;

                    break;

                }

            }

        }

    }

```

```

        // Создаём spin-кнопку для ввода веса

        GtkWidget* spin =
gtk_spin_button_new_with_range(0.0, 1.0, 0.001);

        gtk_spin_button_set_value(GTK_SPIN_BUTTON(spin),
weight);

        gtk_grid_attach(GTK_GRID(grid), spin, j + 1, i +
1, 1, 1);

        spin_buttons[i][j] = spin;

    }

}

// Кнопки внизу

GtkWidget* button_box =
gtk_button_box_new(GTK_ORIENTATION_HORIZONTAL);

    gtk_box_pack_start(GTK_BOX(vbox), button_box, FALSE,
FALSE, 0);

    GtkWidget* save_btn =
gtk_button_new_with_label("Сохранить");

    GtkWidget* cancel_btn =
gtk_button_new_with_label("Заккрыть");

    gtk_box_pack_start(GTK_BOX(button_box), save_btn, TRUE,
TRUE, 5);

    gtk_box_pack_start(GTK_BOX(button_box), cancel_btn, TRUE,
TRUE, 5);

// Передаём данные в обработчик

MatrixData* data = g_new(MatrixData, 1);

data->circles = circles;

data->spin_buttons = spin_buttons;

```

```

        data->node_count = node_count;

        g_signal_connect(save_btn, "clicked",
G_CALLBACK(on_save_matrix_clicked), data);

        g_signal_connect(cancel_btn, "clicked",
G_CALLBACK(gtk_widget_destroy), matrix_window);

        g_signal_connect(matrix_window, "destroy",
G_CALLBACK(gtk_widget_destroyed), &matrix_window);

        gtk_widget_show_all(matrix_window);

    }

    // Обработчик сохранения матрицы

    void on_save_matrix_clicked(GtkWidget* button, gpointer
user_data) {

        MatrixData* data = (MatrixData*)user_data;

        Circles* circles = data->circles;

        int node_count = data->node_count;

        GtkWidget*** spin_buttons = data->spin_buttons;

        // Очищаем существующие связи

        circles->line_count = 0;

        // Создаём новые связи из таблицы

        for (int i = 0; i < node_count; i++) {

            for (int j = 0; j < node_count; j++) {

                double weight =
gtk_spin_button_get_value(GTK_SPIN_BUTTON(spin_buttons[i][j]));

                if (weight > 0.01) {

                    if (circles->line_count < MAX_LINES) {

```

```

        circles->lines[circles-
>line_count].from_index = i;

        circles->lines[circles-
>line_count].to_index = j;

        circles->lines[circles->line_count].weight
= weight;

        circles->line_count++;

    }

}

}

// Освобождаем память
for (int i = 0; i < node_count; i++) {

    g_free(spin_buttons[i]);

}

g_free(spin_buttons);

g_free(data);

// Закрываем окно

gtk_widget_destroy(matrix_window);

matrix_window = NULL;

// Перерисовываем граф

GtkWidget* top =
gtk_widget_get_toplevel(gtk_widget_get_parent(button));

gtk_widget_queue_draw(top);

project_modified = true;

```

```

        g_print("Матрица вероятностей сохранена. Создано %d
связей\n", circles->line_count);

    }

```

					ПГУ 1.090501.12.001 ПЗ	Лист
						135
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Программа оценки вероятностно-временных характеристик
многопроцессорных систем методом стохастических сетей массового
обслуживания
Презентация

					ПГУ 1.090501.12.001 ПЗ	Лист
						136
Изм.	Лист	№ докум.	Подпись	Дата		

Программа оценки вероятностно-временных характеристик многопроцессорных систем методом стохастических сетей массового обслуживания

Петкилева Алёна Алексеевна
Студентка гр. 21ВОЭ1

					ПГУ 1.090501.12.001 ПЗ	Лист
						137
Изм.	Лист	№ докум.	Подпись	Дата		

Актуальность

- Многопроцессорные системы – основа современной вычислительной техники
- При совместной работе процессоры конкурируют за доступ к памяти и контроллерам ввода-вывода
- Возникают очереди и задержки, снижающие производительность
- Необходим инструмент для анализа и оптимизации таких систем

Цель:

Исследование вероятностно-временных характеристик многопроцессорных систем путём разработки программного приложения на основе СеМО.

					ПГУ 1.090501.12.001 ПЗ	Лист
						139
Изм.	Лист	№ докум.	Подпись	Дата		

В процессе разработки были решены следующие задачи:

- Анализ многопроцессорных систем как объектов массового обслуживания
- Разработка аналитических моделей
- Создание приложения с графическим интерфейсом для расчёта характеристик
- Реализация визуального построения графа сети, сохранения и загрузки проектов

Аналоги

- AnyLogic
- GPSS World
- MATLAB + Simulink
- JMT (Java Modelling Tools)

					ПГУ 1.090501.12.001 ПЗ	Лист
						141
Изм.	Лист	№ докум.	Подпись	Дата		

Варианты использования программы

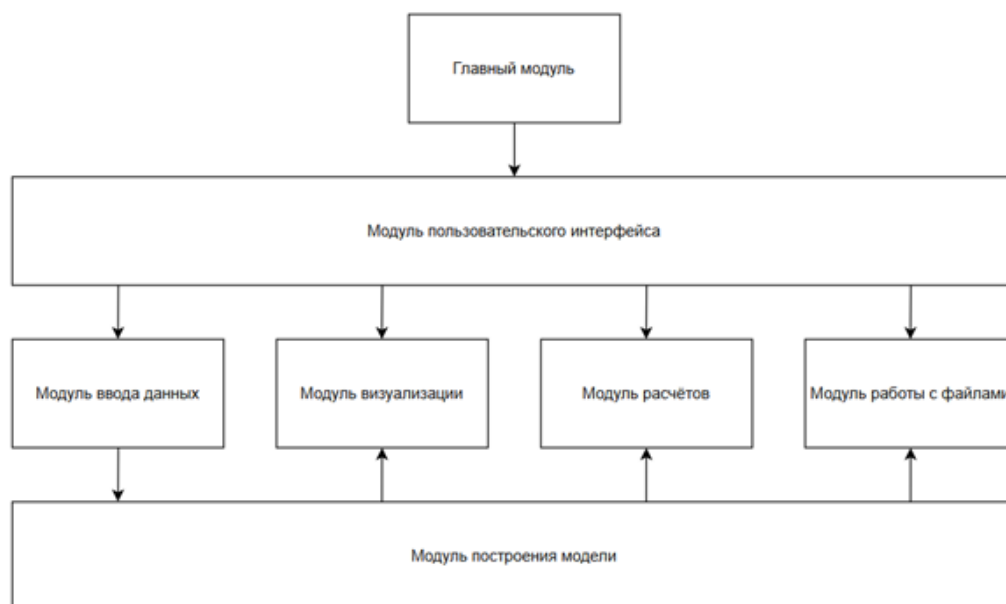


Пользователь



Изм.	Лист	№ докум.	Подпись	Дата

Основные модули программы



Формулы расчёта характеристик СМО

Основные параметры СМО

- λ – интенсивность входящего потока заявок
- μ – интенсивность обслуживания одного канала
- m – количество каналов
- $\rho = \lambda / (m \cdot \mu)$ – коэффициент загрузки

Формулы расчёта характеристик СМО

- **Формула Эрланга (вероятность простоя):**

$$P_0 = \left[\sum_{r=0}^{m-1} \frac{(m\rho)^r}{r!} + \frac{(m\rho)^m}{m! \cdot (1 - \rho)} \right]^{-1}$$

- **Средняя длина очереди:** $L_q = P_0 \cdot \frac{(m\rho)^m \cdot \rho}{m! \cdot (1 - \rho)^2}$

- **Среднее время ожидания в очереди:**

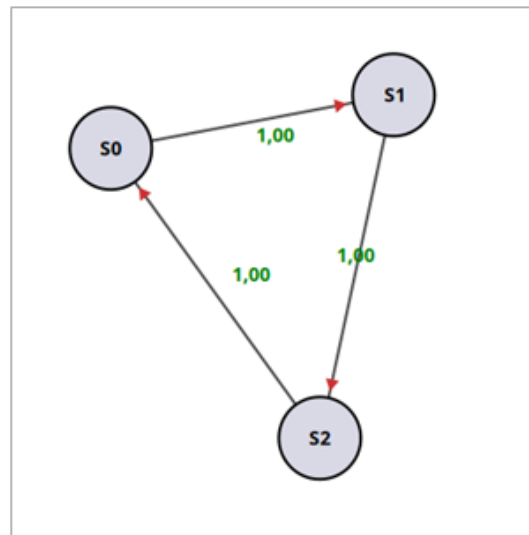
$$W_q = \frac{L_q}{\lambda}$$

- **Среднее время пребывания в системе:**

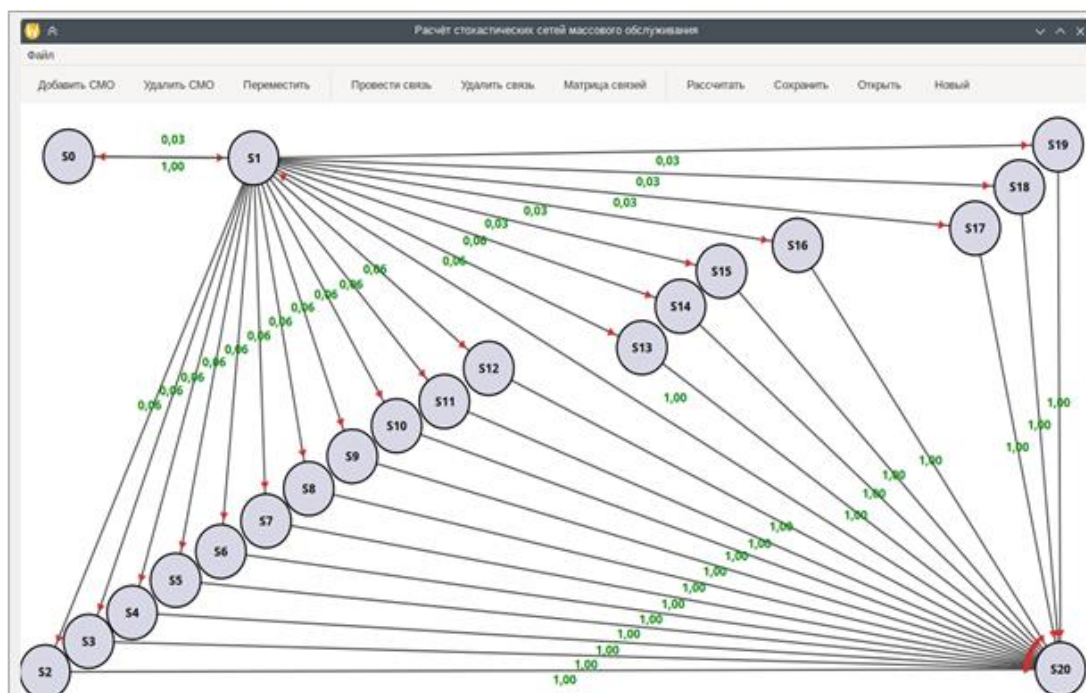
$$W = W_q + \frac{1}{\mu}$$

Визуализация графа сети

- Узлы – круги с подписями $S_0, S_1, S_2...$
- Связи – линии со стрелками (направление)
- Веса связей – вероятности переходов
- Цвета: узлы – серые, линии – тёмно-серые, стрелки – красные, веса – зелёные



Расчёт характеристик



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.12.001 ПЗ

Лист

147

Расчёт характеристик

--- ХАРАКТЕРИСТИКИ ---							
Узел	λ вход	Загрузка	Зан. каналы	L (система)	Lq (очередь)	Wq	W
S0	25,000000	25,000000	0,000000	0,000000	0,000000	0,000000	0,000000
S1	735,287185	0,735287	2,205862	4,606876	2,401014	0,003265	0,006265
S2	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S3	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S4	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S5	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S6	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S7	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S8	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S9	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S10	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S11	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S12	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S13	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S14	45,587789	0,638229	0,638229	1,764180	1,125951	0,024699	0,038699
S15	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777
S16	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777
S17	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777
S18	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777
S19	23,529183	0,470584	0,470584	0,888873	0,418289	0,017777	0,037777
S20	710,291410	0,705218	4,936525	4,946107	0,009581	0,000013	0,006963

Заключение

Разработано:

- Приложение для расчёта характеристик СеМО
- Визуальный редактор графа сети
- Вычислительный модуль для М/М/т и М/М/т/К

Перспективы:

- Режим варьирования
- **Графики зависимостей** – визуализация изменения характеристик от параметров