

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«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 ИМИТАЦИОННОГО
МОДЕЛИРОВАНИЯ
МНОГОПРОЦЕССОРНЫХ ПРИОРИТЕТНЫХ
ПЛАНИРОВЩИКОВ ЗАДАЧ**

Студент

Тишаев 08.06.2026
(подпись, дата)

Тишаев Максим Русланович
(ФИО полностью)

Руководитель

Бикташев 08.06.2026
(подпись, дата)

Бикташев Р. А.
(фамилия, инициалы)

Нормоконтролёр

Бычков 08.06.2026
(подпись, дата)

Бычков А. С.
(фамилия, инициалы)

Рецензент

доцент Пенз ГТУ 16.06.26
(должность, место работы) (подпись, дата)

Петерский С.В.
(фамилия, инициалы)

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

Заведующий кафедрой

Митрохин
(подпись)

М. А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

№ _____)

Секретарь ГЭК

Гурин

Е. И. Гурин

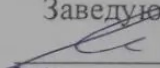
Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 11 » 02 2026 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Тишаев Максим Русланович гр. 21ВОЭ1 факультета вычислительной техники

2. Тема работы Программа имитационного моделирования многопроцессорных приоритетных планировщиков задач

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

3. Руководитель работы к.т.н., доцент кафедры «ВТ» Бикташев Р. А.

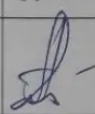
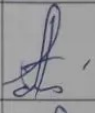
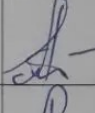
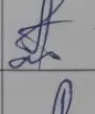
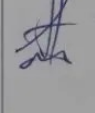
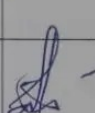
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать VHDL-модель многопроцессорного планировщика задач, обеспечивающую генерацию заявок, их распределение между процессорами, обслуживание в беспriorитетном режиме, а также в режимах относительного и абсолютного приоритета, и сбор статистических характеристик работы системы. В работе необходимо реализовать и сравнить варианты планировщика с глобальной очередью задач и с локальными очередями процессоров.

5. Перечень подлежащих разработке вопросов

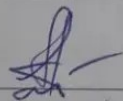
Проанализировать принципы построения многопроцессорных систем обслуживания задач, изучить дисциплины беспriorитетного, относительного и абсолютного приоритета, разработать VHDL-модели планировщиков с глобальной и локальными очередями, выполнить имитационное моделирование их работы и провести сравнение полученных статистических характеристик.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области и анализ существующих аналогов	10	11.02.2026 - 20.02.2026	
2.	Анализ требований и постановка задачи	5	21.02.2026 - 25.02.2026	
3.	Выбор технологий, применяемых при проектировании	15	26.02.2026 - 12.03.2026	
4.	Описание алгоритмов	20	13.03.2026 - 01.04.2026	
5.	Разработка программы для имитационного моделирования многопроцессорных приоритетных планировщиков задач	45	02.04.2026 - 16.05.2026	
6.	Оформление пояснительной записки	5	17.05.2026 - 21.05.2026	

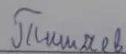
Дата выдачи задания "11" 02 2026 г.

Руководитель работы



(подпись, дата)

Задание к исполнению принял студент



(подпись, дата)

Работу к защите допустить
Декан факультета



16.06.2020

(подпись, дата)

Л. Р. Фионова

Реферат

Пояснительная записка: 155 листов, 59 рисунков, 7 таблиц, 10 источников и 3 приложения.

ПРОГРАММА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ, МНОГОПРОЦЕССОРНАЯ СИСТЕМА, ПЛАНИРОВЩИК ЗАДАЧ, ПРИОРИТЕТНОЕ ПЛАНИРОВАНИЕ, ГЛОБАЛЬНАЯ ОЧЕРЕДЬ, ЛОКАЛЬНЫЕ ОЧЕРЕДИ, ОТНОСИТЕЛЬНЫЕ ПРИОРИТЕТЫ, АБСОЛЮТНЫЕ ПРИОРИТЕТЫ, VHDL, XILINX

Цель работы заключалась в разработке моделей многопроцессорных приоритетных планировщиков задач и имитационном моделировании их работы для получения и сравнения временных характеристик с использованием языка VHDL.

В ходе работы были рассмотрены принципы построения многопроцессорных систем, алгоритмы приоритетного планирования, способы организации глобальной и локальных очередей, а также абсолютный, относительный и беспriorитетный типы обслуживания задач. Разработаны модели планировщика задач с глобальной очередью на 16 мест для 4 процессоров и планировщика задач с 4 локальными очередями по 4 места, каждая из которых соответствует отдельному процессору.

В результате выполнено имитационное моделирование разработанных моделей при различных нагрузках и распределениях заявок по приоритетам. Получены статистические характеристики работы планировщиков, включая количество отказов, среднее время выполнения задачи и среднее время отклика системы. Полученные результаты позволяют сравнить планировщики с глобальной и локальными очередями, а также оценить влияние типа обслуживания на их эффективность.

					ПГУ 1.090501.16.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Тишаев М.Р.		08.06.24			
Пров.		Бикташев Р.А.		08.06.24			
Н. Контр.		Бычков А.С.		08.06.24			
Утв.		Митрохин М.А.		08.06.24			
					Программа имитационного моделирования многопроцессорных приоритетных планировщиков задач		
					Пояснительная записка		
					Литера	Лист	Листов
						2	155
					ФВТ, каф. ВТ гр. 21ВОЭ1		

Содержание

Введение	5
1 Анализ предметной области	7
1.1 Временные показатели существующих систем реального времени	7
1.2 Анализ алгоритмов приоритетного планирования задач	8
1.3 Алгоритмы на основе приоритетных очередей задач	11
1.4 Рассмотрение схем аппаратного приоритетного выбора задач, готовых к выполнению	15
1.5 Анализ методов распределения задач между процессорами	18
1.6 Выбор подхода к разработке многопроцессорной системы	20
1.6.1 Моделирование системы на основе сетей Петри	21
1.6.2 Моделирование системы на языке C++ с использованием библиотеки SystemC	22
1.6.3 Моделирование системы на языке VHDL	24
1.7 Постановка задачи	25
2 Моделирование алгоритмов приоритетного планирования на языке недетерминированных автоматов	27
2.1 Возможности языка НДА для формального описания алгоритмов управления параллельными процессами и ресурсами	27
2.2 Формализация взаимодействия блоков планировщика с глобальной очередью	30
2.3 Формализация взаимодействия блоков планировщика с локальными очередями	34
2.4 Формализация беспriorитетного режима обслуживания	39
2.5 Формализация режима обслуживания с относительными приоритетами	40
2.6 Формализация режима обслуживания с абсолютными приоритетами	42
3 Аппаратная реализация алгоритмов приоритетного планирования	45
3.1 Проблемы управления ресурсами и процессами в многопроцессорных системах	45

3.2	Архитектура аппаратного ядра операционной системы	48
3.3	Моделирование аппаратного планировщика с глобальной очередью	55
3.4	Моделирование аппаратного планировщика с локальными очередями	65
3.5	Получение временных диаграмм и временных характеристик	71
3.6	Выработка критериев сравнительной оценки планировщиков	87
4	Экономика	104
5	Безопасность жизнедеятельности	106
	Заключение.....	109
	Список использованных источников	111
	Приложение А – Схемы программы	113
	Приложение Б – Листинг программы.....	120
	Приложение В – Презентация	148

Введение

Современные вычислительные системы всё чаще строятся как многопроцессорные и параллельные системы обработки информации. Это связано с необходимостью повышения производительности, сокращения времени обработки задач и более эффективного использования вычислительных ресурсов. В таких системах несколько процессоров или вычислительных ядер могут одновременно выполнять различные задачи, что повышает общую пропускную способность системы. Однако увеличение числа процессоров приводит к усложнению управления вычислительным процессом. При построении таких систем важное значение имеют организация взаимодействия процессов, синхронизация их работы и управление доступом к разделяемым ресурсам [1].

Одним из ключевых компонентов многопроцессорной вычислительной системы является планировщик задач. Он определяет порядок выбора заявок на обслуживание и распределяет вычислительную нагрузку между процессорами. От работы планировщика зависит, насколько равномерно используются вычислительные ресурсы, как долго задачи ожидают начала выполнения и насколько эффективно система реагирует на изменение входной нагрузки. Алгоритмы планирования оказывают влияние на время ожидания задач, загрузку процессоров, пропускную способность и общую эффективность функционирования вычислительной системы [2].

Особое значение планирование приобретает в системах реального времени, где результат работы оценивается не только по правильности выполнения задач, но и по времени их завершения. В таких системах задержки при выборе задачи или неравномерное распределение нагрузки могут привести к ухудшению временных характеристик обслуживания. Приоритетное планирование позволяет учитывать различную важность поступающих задач и организовывать их обслуживание с учётом заданных приоритетов [3].

В многопроцессорных системах важным является способ организации очередей. При использовании глобальной очереди все поступающие задачи помещаются в общую очередь, из которой свободные процессоры получают задачи на выполнение. При использовании локальных очередей каждому процессору соответствует собственная очередь задач. Первый подход способствует более равномерному распределению нагрузки, а второй снижает конкуренцию за доступ к общей структуре данных.

Для исследования поведения многопроцессорных систем целесообразно применять имитационное моделирование, позволяющее изучать работу системы при различных значениях параметров, анализировать динамику её функционирования и получать статистические характеристики без создания реального аппаратного устройства. Так как разрабатываемая система ориентирована на аппаратное описание, для её реализации выбран язык VHDL, который используется для описания цифровых устройств и позволяет задавать как структуру аппаратных блоков, так и их поведение во времени [4].

Целью данной работы является разработка и имитационное моделирование многопроцессорных планировщиков задач с глобальной и локальными очередями. В работе рассматриваются шесть моделей планировщиков: с глобальной очередью без приоритетов, с глобальной очередью и относительными приоритетами, с глобальной очередью и абсолютными приоритетами, с локальными очередями без приоритетов, с локальными очередями и относительными приоритетами, а также с локальными очередями и абсолютными приоритетами. Для достижения поставленной цели проводится анализ алгоритмов планирования, разрабатываются VHDL-модели планировщиков, выполняется имитационное моделирование в среде Xilinx 9.1 и сравниваются основные характеристики работы системы: число потерянных заявок, среднее время пребывания заявок в системе, характеристики заявок по каждому приоритету, средняя длина очереди и загрузка процессоров.

1 Анализ предметной области

1.1 Временные показатели существующих систем реального времени

В системах реального времени важную роль играют временные показатели, так как корректность работы таких систем определяется не только правильностью полученного результата, но и временем, за которое этот результат был получен. Вот некоторые из временных характеристик таких систем.

Латентность — это время между моментом поступления запроса на выполнение программы и моментом постановки этого запроса в очередь готовых к выполнению задач. Данная характеристика показывает, насколько быстро система способна принять поступивший запрос и подготовить его к дальнейшему планированию. В системах реального времени латентность имеет большое значение, поскольку задержка на этапе постановки задачи в очередь может повлиять на последующее время ожидания и привести к нарушению временных ограничений.

Время отклика операционной системы реального времени – это время между моментом поступления запроса на выполнение программы и началом его обработки в процессоре. В контексте планирования задач данный показатель характеризует, сколько времени проходит от появления задачи в системе до фактического выделения ей процессорного ресурса. Чем меньше время отклика, тем быстрее система реагирует на поступающие запросы. Для систем реального времени важно не только малое значение времени отклика, но и его предсказуемость.

Джиттером называется отклонение фактического времени реакции, запуска или выполнения задачи от ожидаемого значения. Например, если периодическая задача должна запускаться через равные интервалы времени, но фактически интервалы между запусками изменяются, то возникает джиттер. Эта характеристика показывает стабильность временного поведения системы. Даже

при небольшом среднем времени отклика значительные колебания времени запуска или выполнения задач могут быть недопустимы, особенно в системах управления, связи и обработки сигналов.

Время пребывания задачи в системе – это полный промежуток времени от момента поступления запроса на выполнение программы до момента завершения задачи. Оно включает время постановки запроса в очередь, время ожидания в очереди, время выполнения на процессоре, а также возможные задержки, связанные с вытеснением, переключением задач или ожиданием освобождения вычислительных ресурсов. Данный показатель позволяет оценить, как долго задача находится в системе, и используется для анализа загруженности системы и эффективности работы планировщика.

Дедлайн – это крайний допустимый срок, к которому задача должна быть выполнена. Если задача завершается позже установленного дедлайна, считается, что временное ограничение нарушено. В жестких системах реального времени нарушение дедлайна может привести к отказу системы или опасным последствиям, а в мягких системах реального времени – к снижению качества обслуживания. Поэтому при проектировании планировщиков задач дедлайны являются одним из ключевых факторов, влияющих на выбор алгоритма планирования и оценку его эффективности.

1.2 Анализ алгоритмов приоритетного планирования задач

Приоритетное планирование задач – это способ организации работы вычислительной системы, при котором каждой задаче назначается определенный приоритет. Приоритет показывает степень важности задачи и используется планировщиком при выборе следующей задачи для выполнения. Чем выше приоритет задачи, тем больше вероятность, что она будет выбрана раньше других задач.

В системах реального времени приоритетное планирование имеет особое значение, так как разные задачи могут иметь различную критичность и разные временные ограничения. Например, задачи управления или обработки внешних

событий должны выполняться быстрее, чем менее важные фоновые задачи. Использование приоритетов позволяет упорядочить выполнение задач и обеспечить более быструю обработку наиболее значимых запросов.

В общем виде работа приоритетного планировщика заключается в следующем: после поступления запроса задача помещается в очередь готовых к выполнению задач, затем планировщик анализирует приоритеты задач и выбирает одну из них для передачи на процессор. После завершения выполнения выбранной задачи планировщик снова обращается к очереди и выбирает следующую задачу. Далее рассматриваются некоторые алгоритмы приоритетного планирования.

При планировании с относительными приоритетами диспетчер выбирает из очереди готовых задач задачу с наибольшим приоритетом и передает ее на выполнение процессору. После начала выполнения задача занимает процессор до своего завершения. Если во время ее выполнения в систему поступает новая задача с более высоким приоритетом, текущая задача не прерывается. Новая задача помещается в очередь готовых задач и ожидает освобождения процессора. После завершения текущей задачи диспетчер снова анализирует очередь и выбирает следующую задачу с наибольшим приоритетом. Граф состояний алгоритма представлена на рисунке 1.

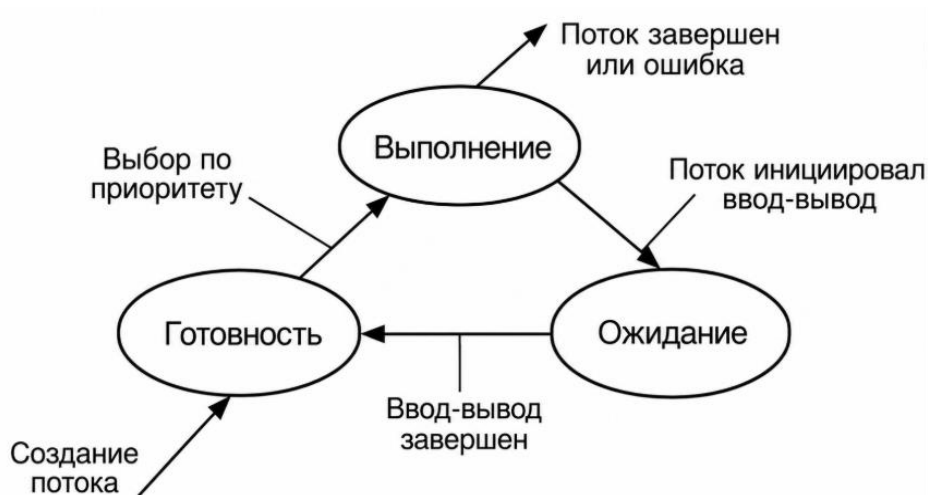


Рисунок 1 – Граф состояний алгоритма обслуживания с относительными приоритетами

При планировании с абсолютными приоритетами диспетчер также выбирает из очереди готовых задач задачу с наибольшим приоритетом и передает ее на выполнение процессору. Однако если во время выполнения текущей задачи поступает новая задача с более высоким приоритетом, диспетчер проверяет наличие свободного процессора. Если свободный процессор отсутствует, выполнение текущей менее приоритетной задачи прерывается, ее состояние сохраняется, а процессор передается задаче с более высоким приоритетом. Прерванная задача возвращается в очередь готовых задач и может быть продолжена позднее. Граф состояний алгоритма представлена на рисунке 2.

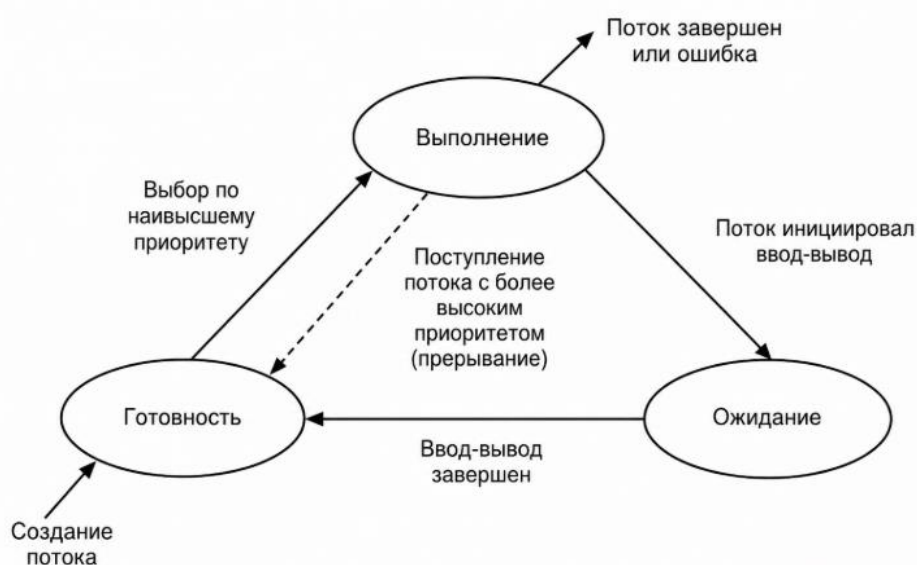


Рисунок 2 – Граф состояний алгоритма обслуживания с абсолютными приоритетами

При планировании с динамическими приоритетами диспетчер выбирает задачу на основе текущего значения ее приоритета, которое может изменяться во время нахождения задачи в системе. Например, приоритет может повышаться при длительном ожидании в очереди или при приближении дедлайна. После поступления задачи ей назначается начальный приоритет, затем в процессе работы системы значения приоритетов пересчитываются по заданному правилу. Диспетчер выбирает задачу с наибольшим текущим приоритетом и передает ее на выполнение. Такой подход позволяет учитывать изменение состояния системы и

более гибко управлять порядком выполнения задач. Граф состояний алгоритма представлена на рисунке 3.

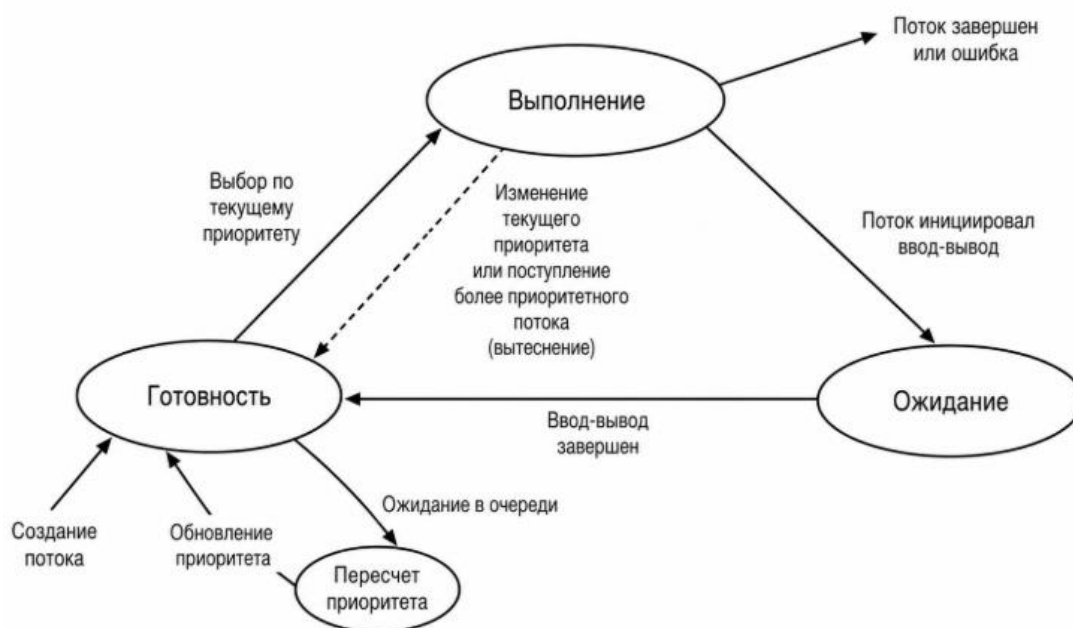


Рисунок 3 – Граф состояний алгоритма обслуживания с динамическими приоритетами

1.3 Алгоритмы на основе приоритетных очередей задач

При рассмотрении алгоритмов планирования в многопроцессорных системах необходимо определить, какой объект будет передаваться на выполнение: процесс или поток. В ранних вычислительных системах основным объектом планирования являлся процесс, так как процессы, как правило, были однопоточными. В современных операционных системах один процесс может содержать несколько потоков выполнения, поэтому планировщик должен учитывать не только процессы, но и отдельные потоки.

Если потоки реализованы на уровне пользовательского пространства и операционная система не имеет информации об их существовании, то планирование фактически выполняется на уровне процессов. В этом случае ядро выбирает процесс, а распределение выполнения между потоками внутри него может выполняться пользовательской библиотекой. Если же потоки

поддерживаются ядром, то каждый поток рассматривается как самостоятельный объект планирования. В таком случае диспетчер может выбирать для выполнения конкретный поток, а принадлежность потока к определенному процессу играет второстепенную роль.

В однопроцессорной системе задача планировщика заключается в выборе следующего потока для выполнения. В многопроцессорной системе задача усложняется, так как необходимо определить не только какой поток должен быть запущен, но и на каком процессоре он будет выполняться. Это особенно важно при моделировании многопроцессорного приоритетного планировщика, поскольку от способа распределения потоков зависит загрузка процессоров и время ожидания задач.

Одним из распространенных подходов является использование единой очереди готовых потоков для всей системы. В такой очереди находятся все потоки, готовые к выполнению, но еще не получившие процессорное время. Очередь может быть представлена как общий список или как набор очередей для различных уровней приоритета. Организация общей структуры данных представлена на рисунке 4.

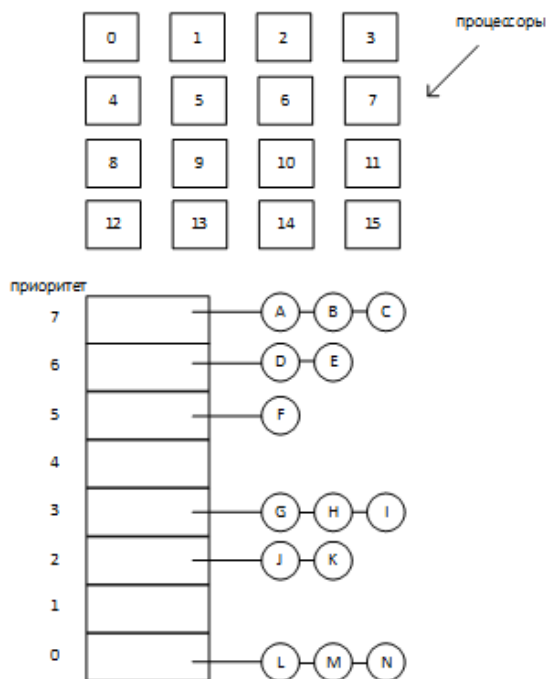


Рисунок 4 – Единая структура данных для планирования работы

При освобождении одного из процессоров диспетчер обращается к общей очереди готовых потоков и выбирает поток с наивысшим приоритетом. После этого выбранный поток передается на свободный процессор для выполнения. Ситуация, при которой освободившийся процессор выбирает из общей очереди поток с наивысшим приоритетом, представлена на рисунке 5.

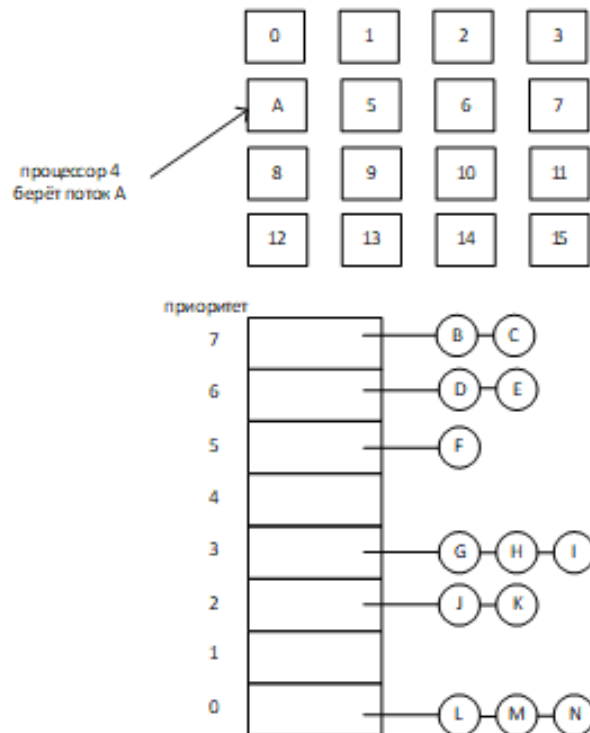


Рисунок 5 – Блокировка свободного процессора процессом с наивысшим приоритетом

После этого при освобождении следующего процессора из очереди выбирается очередной поток с максимальным приоритетом среди оставшихся. Таким образом, потоки поочередно распределяются между свободными процессорами в соответствии с их приоритетами. Последующее назначение потока на освободившийся процессор представлено на рисунке 6.

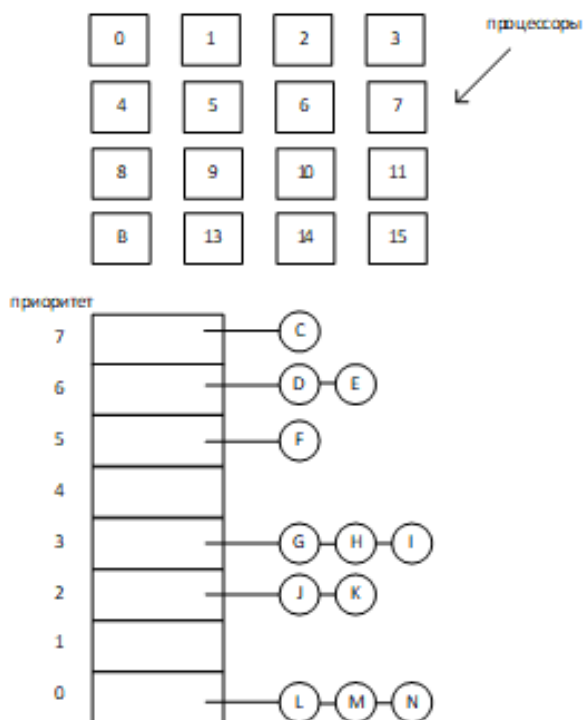


Рисунок 6 – Последующая блокировка свободного процессора

Использование общей приоритетной очереди позволяет более равномерно распределять нагрузку между процессорами. Если один процессор освобождается раньше других, он может взять следующий готовый поток из общей очереди, поэтому снижается вероятность ситуации, когда один процессор простаивает, а другой перегружен. Такой подход удобен для независимых потоков, которые не требуют выполнения на строго определенном процессоре.

Однако единая очередь готовых потоков имеет и недостатки. При увеличении числа процессоров возрастает количество обращений к общей структуре данных, что может приводить к конкуренции за доступ к очереди. Кроме того, при блокировке потока, ожидании ввода-вывода или переключении контекста возникают дополнительные накладные расходы. Поэтому при разработке многопроцессорных планировщиков важно учитывать не только правило выбора потока по приоритету, но и способ организации очередей готовности.

1.4 Рассмотрение схем аппаратного приоритетного выбора задач, готовых к выполнению

В данном подразделе рассматриваются аппаратные способы выбора готовой к выполнению задачи с наивысшим приоритетом. Так как планировщик ориентирован на системы реального времени, он должен обеспечивать быстрое и предсказуемое принятие решения при возникновении событий: завершении задачи, блокировке, освобождении процессора или поступлении новой задачи.

Приоритеты задач могут назначаться статически или динамически. При статическом подходе приоритет каждой задачи задается заранее и не изменяется во время выполнения. Такая схема проще для аппаратной реализации, так как планировщику необходимо только определить среди готовых задач задачу с наибольшим приоритетом.

При динамическом подходе приоритет может изменяться во время работы системы в зависимости от временных параметров задачи, например дедлайна, времени поступления или времени выполнения. Такой вариант более гибкий, но требует дополнительной логики для пересчета приоритетов и усложняет аппаратную структуру.

В рамках проекта в качестве базового варианта целесообразно использовать схему со статическими фиксированными приоритетами. Это позволяет упростить разработку аппаратного планировщика и сосредоточиться на механизме приоритетного выбора и распределения задач между процессорами. При этом структура планировщика может быть расширена в дальнейшем для поддержки динамического назначения приоритетов.

Далее рассматриваются три варианта схем: схема приоритетного выбора «полное дерево», схема последовательного приоритетного выбора и схема гибридного приоритетного выбора.

В схеме «полное дерево» выбор готовой задачи с наивысшим приоритетом выполняется с помощью набора сравнительных блоков, соединенных в древовидную структуру. На первом уровне схемы попарно сравниваются

приоритеты готовых задач. Затем результаты этих сравнений передаются на следующий уровень, где снова выбирается задача с более высоким приоритетом. Процесс продолжается до тех пор, пока на выходе схемы не останется одна задача, имеющая наибольший приоритет среди всех готовых к выполнению задач. Такая организация позволяет выполнять часть сравнений параллельно, за счет чего уменьшается время выбора задачи. Недостатком схемы является увеличение количества аппаратных элементов при росте числа задач. Схема приоритетного выбора типа «полное дерево» представлена на рисунке 7.

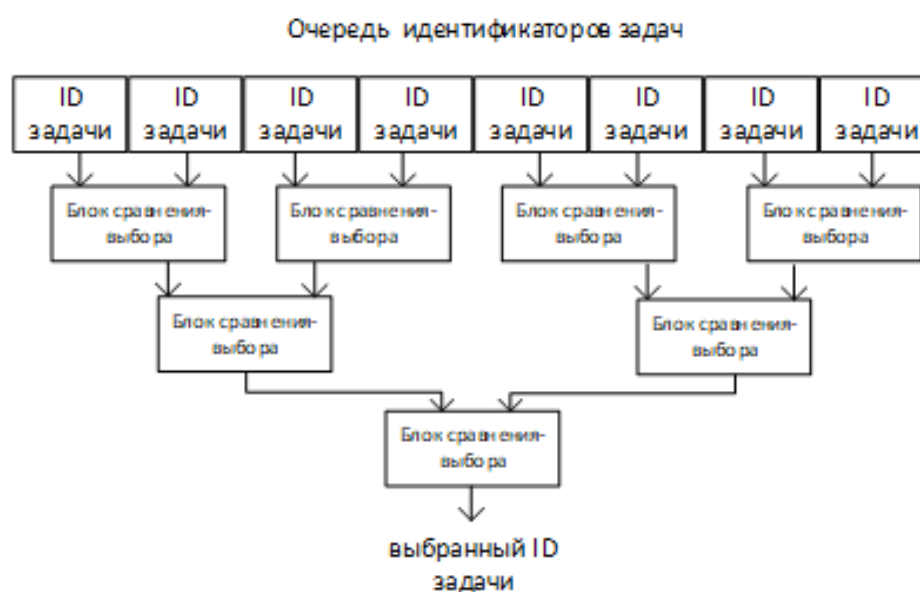


Рисунок 7 – Схема приоритетного выбора «полное дерево»

В схеме последовательного приоритетного выбора задачи анализируются поочередно. На каждом шаге текущая задача сравнивается с уже выбранным кандидатом на выполнение. Если приоритет текущей задачи выше, она становится новым кандидатом. После проверки всех готовых задач на выходе схемы формируется номер задачи с наивысшим приоритетом. Такой подход проще с точки зрения аппаратной реализации, так как требует меньше сравнительных блоков и управляющей логики. Однако время выбора зависит от количества задач: чем больше задач необходимо проверить, тем дольше формируется результат. Поэтому последовательная схема удобна для систем с небольшим числом задач или

в случаях, когда минимизация аппаратных затрат важнее максимальной скорости выбора. Схема последовательного приоритетного выбора представлена на рисунке 8.

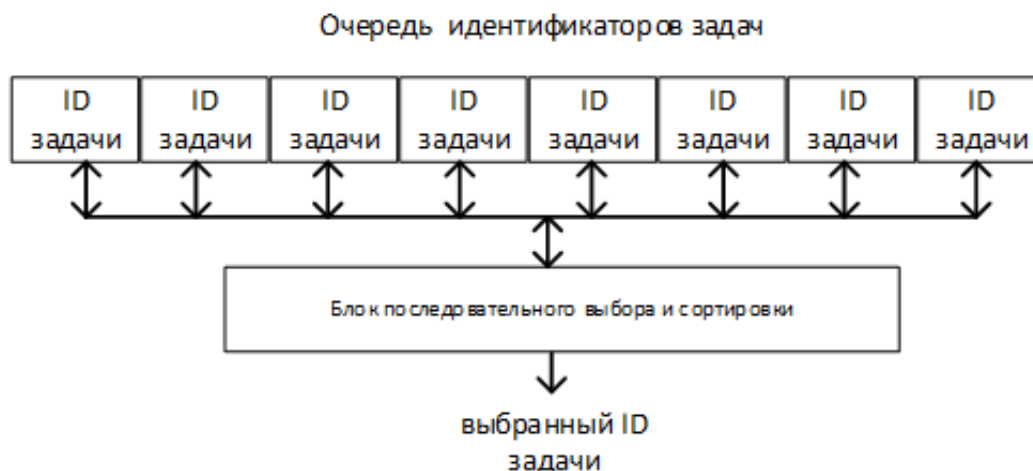


Рисунок 8 – Схема последовательного приоритетного выбора

Гибридная схема объединяет принципы полного дерева и последовательного выбора. В таком варианте все готовые задачи могут быть разделены на несколько групп. Внутри каждой группы выбор наиболее приоритетной задачи выполняется последовательным способом или с помощью небольшой локальной схемы сравнения. Затем результаты, полученные от отдельных групп, передаются на общий блок выбора, который определяет задачу с наивысшим приоритетом среди групповых кандидатов. Такой подход позволяет уменьшить аппаратные затраты по сравнению с полным деревом и одновременно сократить время выбора по сравнению с полностью последовательной схемой. Гибридная схема является компромиссным вариантом между быстродействием и сложностью аппаратной реализации. Гибридная схема приоритетного выбора представлена на рисунке 9.

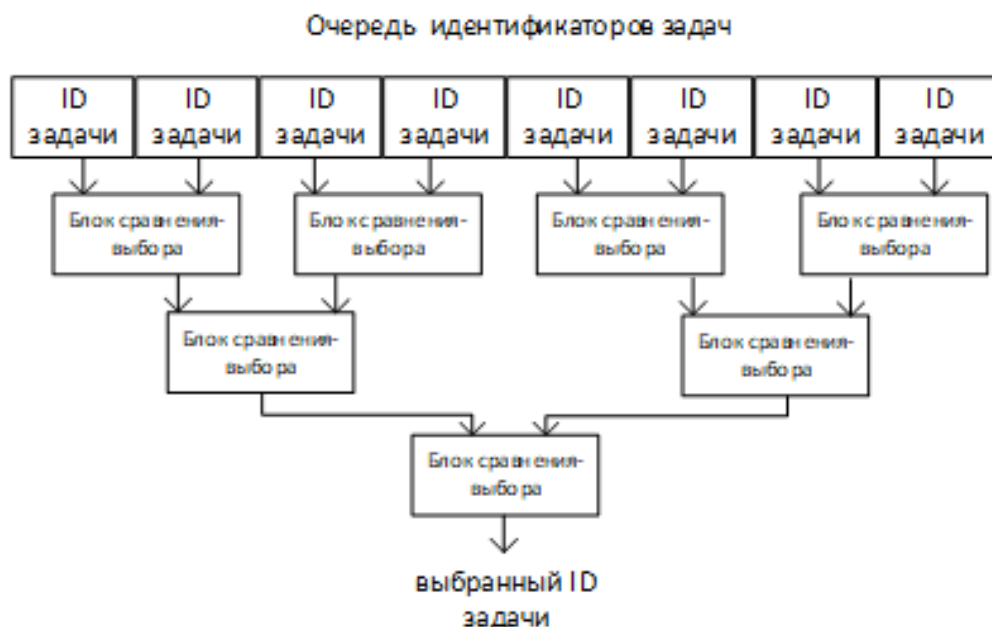


Рисунок 9 – Схема гибридного приоритетного выбора

1.5 Анализ методов распределения задач между процессорами

В многопроцессорных системах планировщик должен решать не только задачу выбора следующей готовой к выполнению задачи, но и задачу ее распределения между доступными процессорами. В отличие от однопроцессорной системы, где выполнение возможно только на одном процессоре, в многопроцессорной системе несколько задач могут выполняться параллельно. Поэтому от выбранного метода распределения зависит загрузка процессоров, время ожидания задач и общая эффективность работы планировщика.

При распределении задач необходимо учитывать состояние процессоров, наличие готовых задач, их приоритеты и возможную неравномерность нагрузки. Если распределение организовано неэффективно, может возникнуть ситуация, при которой один процессор перегружен, а другой простаивает. Для решения этой задачи обычно применяются два основных подхода: использование глобальной очереди задач и использование локальных очередей процессоров.

Глобальная очередь задач предполагает, что все готовые к выполнению задачи помещаются в одну общую очередь. Когда один из процессоров освобождается, планировщик обращается к глобальной очереди, выбирает задачу с

наибольшим приоритетом и передает ее на выполнение свободному процессору. Такой подход удобен тем, что все процессоры работают с единым набором задач, поэтому нагрузка распределяется более равномерно. Схема распределения задач с использованием глобальной очереди представлена на рисунке 10.

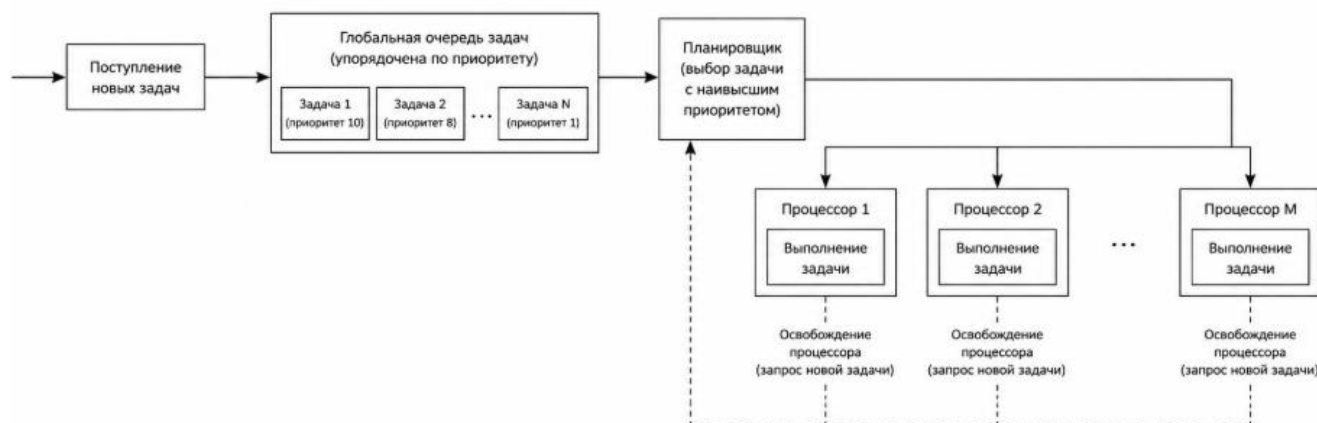


Рисунок 10 – Распределение задач с использованием глобальной очереди

Недостатком глобальной очереди является возможная конкуренция за доступ к общей структуре данных. При увеличении числа процессоров несколько процессоров могут одновременно обращаться к одной очереди, что требует синхронизации и может увеличивать накладные расходы.

Локальные очереди задач предполагают, что у каждого процессора есть собственная очередь готовых задач. После поступления задача назначается конкретному процессору и помещается в его локальную очередь. Далее каждый процессор выбирает задачи только из своей очереди. Такой подход снижает нагрузку на общий планировщик и уменьшает конкуренцию за доступ к очереди, так как каждый процессор работает со своей структурой данных. Схема распределения задач с использованием локальных очередей представлена на рисунке 11.



Рисунок 11 – Распределение задач с использованием локальных очередей

Однако локальные очереди могут приводить к неравномерной загрузке процессоров. Например, очередь одного процессора может быть перегружена, в то время как очередь другого процессора уже пуста. В таком случае требуется дополнительный механизм балансировки нагрузки или переноса задач между очередями.

Таким образом, глобальная очередь проще обеспечивает равномерное распределение задач между процессорами, но может стать узким местом при большом числе обращений. Локальные очереди уменьшают конкуренцию за общую очередь, но требуют дополнительных средств балансировки нагрузки. Выбор подхода зависит от требований к быстродействию, сложности реализации и особенностей моделируемого многопроцессорного планировщика.

1.6 Выбор подхода к разработке многопроцессорной системы

Для дальнейшей разработки многопроцессорной системы необходимо выбрать подход к ее моделированию, который позволит описать структуру планировщика, поведение задач, работу очередей и взаимодействие нескольких процессоров. Поэтому далее рассматриваются возможные способы разработки многопроцессорной системы

1.6.1 Моделирование системы на основе сетей Петри

Сети Петри применяются для описания систем, в которых важны состояния, события, параллельное выполнение и конкуренция за общие ресурсы. Такой подход особенно полезен, когда необходимо не просто воспроизвести работу системы, а понять её внутреннюю логику: какие состояния возможны, какие события могут произойти, при каких условиях они выполняются и к чему приводят.

Модель на основе сети Петри строится как граф. В нём есть позиции, переходы и маркеры. Позиции показывают возможные состояния системы или наличие определённых условий. Маркеры находятся в позициях и отражают текущее состояние модели. Переходы соответствуют событиям или действиям. Если все необходимые условия выполнены, переход становится доступным для срабатывания. После его срабатывания маркеры перемещаются из одних позиций в другие, и состояние системы изменяется.

При построении такой модели сначала выделяются основные состояния объекта. Например, для системы с задачами это могут быть состояния ожидания, готовности к выполнению, выполнения, завершения или блокировки. Затем определяются события, которые переводят систему из одного состояния в другое: поступление задачи, выбор ресурса, начало обработки, завершение операции, освобождение ресурса. В результате получается модель, где поведение системы выражается не через программные инструкции, а через правила изменения состояний.

Сильная сторона сетей Петри заключается в том, что они показывают параллельность. Если в модели несколько переходов доступны одновременно и не мешают друг другу, это означает, что соответствующие действия могут выполняться независимо. Если же несколько переходов претендуют на один и тот же маркер, возникает конфликт. Поэтому сети Петри хорошо подходят для анализа систем, где несколько процессов взаимодействуют, ожидают ресурсы или конкурируют между собой.

Для моделей планирования такой подход особенно удобен. Задачи можно представить маркерами, ресурсы – отдельными позициями, а действия

планировщика – переходами. Например, задача может перейти из состояния ожидания в состояние выполнения только при наличии свободного ресурса. Если ресурсов несколько, сеть позволяет наглядно показать, как задачи распределяются между ними и в каких ситуациях возможно одновременное выполнение.

Важное преимущество сетей Петри – возможность формального анализа. По построенной модели можно проверять, достижимы ли определённые состояния, возможны ли тупики, не возникает ли взаимная блокировка, не остаются ли задачи навсегда в ожидании. Это отличает сети Петри от обычной имитационной модели: они позволяют не только запускать сценарии, но и исследовать свойства системы более строго.

При необходимости модель можно расширять. В простых сетях маркеры одинаковые, но в цветных сетях Петри им можно задавать типы и параметры. Это позволяет различать задачи, приоритеты, классы ресурсов или разные виды событий. Во временных сетях Петри добавляются задержки, благодаря чему можно учитывать длительность операций и анализировать поведение системы во времени.

Главное ограничение такого подхода состоит в том, что сеть Петри обычно не является готовым аппаратным или программным описанием. Она служит формальной моделью поведения. После её построения и проверки полученные решения можно перенести в SystemC-модель, программный алгоритм или описание аппаратного блока на VHDL. Поэтому сети Петри целесообразно использовать на ранних этапах, когда необходимо разобраться в логике взаимодействий, найти проблемные состояния и убедиться в корректности общей схемы работы системы.

1.6.2 Моделирование системы на языке C++ с использованием библиотеки SystemC

Разработка на SystemC в первую очередь связана с созданием исполняемой имитационной модели системы. Такая модель нужна для того, чтобы заранее проверить поведение проектируемого объекта, оценить взаимодействие его частей и рассмотреть выбранные архитектурные решения до перехода к детальному аппаратному описанию. В отличие от низкоуровневого описания аппаратуры,

SystemC позволяет сначала рассматривать систему на более общем уровне: через модули, процессы, события, каналы связи и модельное время.

Основой разработки является разбиение системы на отдельные взаимодействующие компоненты. Каждый компонент оформляется как модуль SystemC и отвечает за определённую часть поведения. Между модулями задаётся обмен данными через сигналы, события, FIFO-каналы, пользовательские каналы или транзакции. Благодаря этому взаимодействие необязательно сразу описывать через отдельные управляющие линии и каждый бит интерфейса: его можно представить более крупными операциями, например передачей команды, сообщения, заявки или набора данных.

После определения структуры модели описываются данные, состояния и поведение компонентов. Данные можно представлять в виде структур, классов или сообщений, что удобно при описании сложных объектов. Поведение задаётся процессами SystemC, которые выполняются параллельно под управлением имитационного ядра. Эти процессы реагируют на события, ожидают изменения сигналов, обмениваются данными и учитывают задержки модельного времени. За счёт этого можно описывать одновременную работу частей системы без преждевременного перехода к детальной аппаратной схеме.

Важной частью разработки является тестовое окружение. Оно формирует входные воздействия, задаёт сценарии работы и позволяет наблюдать за поведением модели. В этой части допускается использовать более свободные средства C++: контейнеры, файлы, случайную генерацию данных и вспомогательные функции, поскольку тестовое окружение не является частью будущей аппаратной реализации. После этого модель компилируется как обычная программа на C++ с подключением библиотеки SystemC и запускается на симуляцию, во время которой имитационное ядро управляет процессами, событиями и модельным временем.

Главное достоинство такого подхода – возможность ранней проверки системы без необходимости сразу описывать её на уровне регистров, тактов и управляющих сигналов. Сначала можно проверить общую логику работы, сравнить

несколько вариантов архитектуры, выявить ошибки во взаимодействии компонентов и постепенно уточнить модель: добавить более точные интерфейсы, фиксированные размеры данных, тактовую синхронизацию и другие ограничения, приближающие её к аппаратному описанию.

В результате на SystemC создаётся имитационная модель, которая используется как основа для проверки поведения системы. После уточнения такая модель может стать основой для дальнейшего получения VHDL-кода вручную или с помощью средств высокоуровневого синтеза. Полученное аппаратное описание затем можно использовать для синтеза в цифровую схему.

1.6.3 Моделирование системы на языке VHDL

Разработка модели на VHDL связана с описанием цифровой системы в форме, близкой к её будущей аппаратной реализации. Такой подход используется тогда, когда важно не только проверить поведение устройства, но и получить основу для дальнейшего синтеза цифровой схемы.

Работа над моделью начинается с определения объекта проектирования и его внешнего интерфейса. Необходимо описать, какие входные и выходные сигналы имеет устройство, какие данные оно принимает, какие результаты формирует и каким образом взаимодействует с другими блоками. В VHDL это обычно выражается через описание сущности устройства, где задаются порты и внешние связи модели.

После этого описывается внутренняя архитектура устройства. Она может быть поведенческой, структурной или смешанной. Поведенческое описание показывает, как устройство реагирует на входные сигналы и как изменяет свои состояния. Структурное описание, наоборот, делает акцент на составе устройства: какие компоненты входят в модель и как они соединены между собой.

Особое значение имеют сигналы, процессы и типы данных. Сигналы используются для передачи значений между частями модели, процессы позволяют описывать параллельную работу устройства, а типы данных задают форму представления информации. При этом заранее учитываются разрядность сигналов, допустимые значения, условия синхронизации и правила изменения состояний.

Модель на VHDL обычно требует более строгого описания, чем высокоуровневая имитационная модель. Здесь важно не только указать, что должно происходить, но и представить это в форме, пригодной для цифровой схемы. Поэтому при разработке учитываются тактовые сигналы, сброс, регистры, комбинационная логика, конечные автоматы и связи между блоками.

После написания описания выполняется моделирование работы устройства. Для этого создаётся тестовое окружение, которое подаёт входные воздействия и позволяет проверить реакцию модели. На этом этапе можно обнаружить ошибки в логике работы, неправильные переходы состояний, некорректные значения сигналов или нарушения во взаимодействии блоков.

Одно из главных достоинств VHDL заключается в том, что модель может служить не только для проверки поведения, но и для дальнейшего синтеза. Если описание написано в синтезируемом стиле, его можно преобразовать в аппаратную структуру: регистры, логические элементы, соединения и управляющие схемы.

Таким образом, разработка на VHDL строится вокруг точного описания цифрового устройства: его интерфейса, внутренней архитектуры, сигналов, процессов и правил изменения состояний.

1.7 Постановка задачи

Целью работы является имитационное моделирование многопроцессорных приоритетных планировщиков задач на языке VHDL. Выбор VHDL обусловлен необходимостью аппаратно-ориентированного описания планировщика, возможностью последующего снятия характеристик в реальной цифровой системе, а также использованием ранее разработанного кода, на основе которого продолжается данный проект.

В рамках работы необходимо разработать два варианта планировщика: планировщик с глобальной очередью на 16 мест для 4 процессоров и планировщик с 4 локальными очередями, каждая из которых содержит 4 места и соответствует одному из 4 процессоров.

Для каждого варианта необходимо реализовать три режима обслуживания задач: абсолютный приоритет, относительный приоритет и беспriorитетный режим.

Разработанные модели должны обеспечивать возможность задания параметров моделирования: интенсивности поступления заявок, времени такта, общего времени моделирования, числа приоритетов, а также процентного соотношения заявок по каждому приоритету. Это необходимо для проведения экспериментов при различных нагрузках и разных распределениях входного потока задач.

Модели должны обеспечивать прием входных задач, размещение их в очередях, выбор задачи на выполнение, назначение задачи на процессор, обработку освобождения процессора и фиксацию отказов при невозможности постановки задачи в очередь.

В ходе моделирования необходимо определить следующие характеристики: общее количество отказов, количество отказов по приоритетам, среднее время выполнения задачи в системе, среднее время выполнения задачи в системе по каждому приоритету, среднее время отклика системы, среднее время отклика системы по каждому приоритету. Полученные результаты должны позволить сравнить планировщики с глобальной и локальными очередями, а также оценить влияние режимов обслуживания и параметров входного потока на работу многопроцессорных приоритетных планировщиков.

2 Моделирование алгоритмов приоритетного планирования на языке недетерминированных автоматов

2.1 Возможности языка НДА для формального описания алгоритмов управления параллельными процессами и ресурсами

При разработке моделей многопроцессорных планировщиков задач необходимо формально описать порядок возникновения и обработки событий, связанных с поступлением задач, постановкой их в очередь, выбором задачи для обслуживания, назначением процессора, выполнением задачи и фиксацией статистических характеристик. Так как указанные процессы могут протекать параллельно и зависят от состояния общих ресурсов, для их описания целесообразно использовать аппарат недетерминированных автоматов.

Недетерминированный автомат позволяет представить функционирование системы в виде совокупности частных событий и логических условий их возникновения. В рамках данной работы под частным событием понимается элементарное действие или состояние модели, например поступление новой задачи, наличие свободного места в очереди, запись задачи в очередь, отказ в постановке задачи, готовность задачи к обслуживанию, загрузка задачи в процессор, выполнение задачи и завершение обслуживания.

Для описания алгоритмов планирования используются следующие обозначения:

$S_{i(t)}$ – i -е частное событие НДА, рассматриваемое в текущий момент времени t ;

$S_{j(t+1)}$ – частное событие НДА, возникающее на следующем такте моделирования;

$x_{ij(t)}$ – логическое условие перехода от частного события S_i к частному событию S_j в момент времени t ;

$Y_{j(t)}$ – выходное действие или управляющий сигнал, формируемый при возникновении частного события S_j ;

$V_{j(t)}$ – информационный параметр задачи или блока модели, связанный с j -м частным событием;

t – текущий такт моделирования;

$t + 1$ – следующий такт моделирования.

Частные события в дальнейшем обозначаются с использованием верхних и нижних индексов. Верхний индекс t используется для событий, связанных с задачей или очередью. Верхний индекс p используется для событий, связанных с процессором. Обозначение Q_j применяется для событий, относящихся к j -й локальной очереди. Индекс j задаёт номер процессора или номер локальной очереди.

Например:

S_i^t – ожидание поступления новой задачи;

S_Q^t – нахождение задачи в очереди;

S_{Aj}^p – выполнение задачи j -м процессором.

Переходы между частными событиями описываются с помощью канонических уравнений НДА. В общем виде такое уравнение (1) задаёт условие возникновения некоторого события на следующем такте моделирования:

$$S_{j(t+1)} = F(S_{i(t)}, x_{ij(t)}, Y_{j(t)}), \quad (1)$$

где $S_{j(t+1)}$ – частное событие, возникающее на следующем такте моделирования;

$S_{i(t)}$ – частное событие или совокупность частных событий на текущем такте;

$x_{ij(t)}$ – логическое условие перехода;

$Y_{j(t)}$ – управляющее действие, связанное с возникновением события S_j .

При записи канонических уравнений НДА используются следующие логические операции:

$A \cdot B$ – логическая операция И, означающая одновременное выполнение условий A и B ;

$A \vee B$ – логическая операция ИЛИ, означающая выполнение хотя бы одного из условий А или В;

$\neg A$ – логическая операция НЕ, означающая отрицание условия А.

Таким образом, каждое каноническое уравнение НДА описывает не отдельную строку программного кода, а определённый механизм функционирования модели. Например, механизм записи поступившей задачи в очередь можно представить следующим уравнением (2):

$$S_{WR}^{t(t+1)} = S_i^{t(t)} \cdot S_{FQ}^{t(t)}, \quad (2)$$

где $S_{WR}^{t(t+1)}$ – событие записи задачи в очередь на следующем такте моделирования;

$S_i^{t(t)}$ – событие поступления новой задачи на текущем такте;

$S_{FQ}^{t(t)}$ – событие наличия свободного места в очереди на текущем такте.

Данное уравнение означает, что событие записи задачи в очередь возникает на следующем такте в том случае, если на текущем такте поступила новая задача и в очереди имеется свободное место.

Если свободного места в очереди нет, возникает событие отказа в постановке задачи, уравнение (3):

$$\neg RF^{t(t+1)} = S_i^{t(t)} \cdot \neg S_{FQ}^{t(t)}, \quad (3)$$

где $\neg RF^{t(t+1)}$ – событие отказа в постановке задачи в очередь на следующем такте моделирования;

$S_i^{t(t)}$ – событие поступления новой задачи на текущем такте;

$\neg S_{FQ}^{t(t)}$ – условие отсутствия свободного места в очереди.

Аналогично могут быть описаны механизмы выбора задачи из очереди, загрузки задачи в процессор, выполнения задачи, завершения обслуживания и вытеснения задачи в режиме абсолютных приоритетов.

В разрабатываемой модели канонические уравнения НДА используются для описания взаимодействия основных блоков планировщика. К таким блокам относятся генератор задач, очередь задач, блок выбора задачи, блок назначения процессора, процессоры и блок сбора статистики. Для моделей с локальными

очередями дополнительно описывается механизм выбора локальной очереди для размещения поступившей задачи.

Использование НДА позволяет представить алгоритм работы планировщика в формализованном виде: сначала выделяются частные события, затем строится граф функционирования модели, после чего для основных переходов составляется система канонических уравнений. Такой подход обеспечивает связь между алгоритмическим описанием планировщика и его дальнейшей реализацией на языке VHDL.

2.2 Формализация взаимодействия блоков планировщика с глобальной очередью

В модели планировщика с глобальной очередью все поступающие задачи размещаются в общей очереди, из которой затем выбираются для передачи свободным процессорам. Такая структура позволяет централизованно управлять потоком задач и распределять их между процессорными блоками. Основными блоками модели являются генератор задач, блок управления глобальной очередью, глобальная очередь, блок выбора процессора, блок синхронизации назначения, процессорные блоки и блок сбора статистики.

Обобщённая последовательность обработки задачи в модели с глобальной очередью включает поступление новой задачи, проверку наличия свободного места в очереди, запись задачи в очередь или отказ в постановке, выбор задачи из очереди, назначение свободного процессора, загрузку задачи в процессор, выполнение задачи и фиксацию завершения обслуживания.

Для формального описания данного процесса вводятся частные события и условия переходов НДА, представленные в таблице 1.

Таблица 1 – Обозначения частных событий и условий переходов НДА для модели планировщика с глобальной очередью

Обозначение	Описание	Сигналы в модели
S_i^t	Ожидание поступления новой задачи	S_task_arrived
S_{FQ}^t	Наличие свободного места в глобальной очереди	$\neg$ S_queue_full
S_Q^t	Нахождение задачи в глобальной очереди	$\neg$ S_queue_empty, V_queue_length > 0
S_{WR}^t	Запись задачи в глобальную очередь	S_new_wrreq
S_{RF}^t	Отказ в постановке задачи в очередь	S_refuse
S_{CR}^t	Фиксация факта создания задачи	S_task_created
S_{GPK}^t	Готовность задачи из глобальной очереди к передаче процессору	S_top_valid_k
S_{SLj}^p	Выбор процессора с номером j для обслуживания задачи	S_assign_valid_k, V_assign_cpu_k
S_{OFk}^t	Удаление выбранной задачи из глобальной очереди	S_pop_top_k
S_{LTj}^p	Загрузка задачи в процессор с номером j	S_load_task
S_{Bj}^p	Занятость процессора с номером j	S_cpu_busy
S_{Aj}^p	Выполнение задачи процессором с номером j	S_cpu_running
S_{Ej}^p	Завершение выполнения задачи процессором с номером j	S_task_done
$x_{top_{k(t)}}$	Условие наличия допустимой задачи в позиции k глобальной очереди	S_top_valid_k
$x_{sel_{j(t)}}$	Условие выбора процессора с номером j блоком назначения	V_assign_cpu_k = j
$x_{done_{j(t)}}$	Условие окончания времени обработки задачи процессором с номером j	S_task_done

В таблице 1 индекс k используется для обозначения позиции задачи, выдаваемой из глобальной очереди, а индекс j – для обозначения номера

процессора, которому назначается выбранная задача. Такое разделение связано с тем, что в модели сначала определяется готовая к передаче задача из очереди, после чего для неё выбирается свободный процессор.

На основании выделенных частных событий строится граф функционирования планировщика с глобальной очередью, представленный на рисунке 12. Он отражает основные переходы между событиями: поступление задачи, запись в очередь или отказ, выбор задачи из очереди, назначение процессору, загрузку, выполнение и завершение обслуживания.

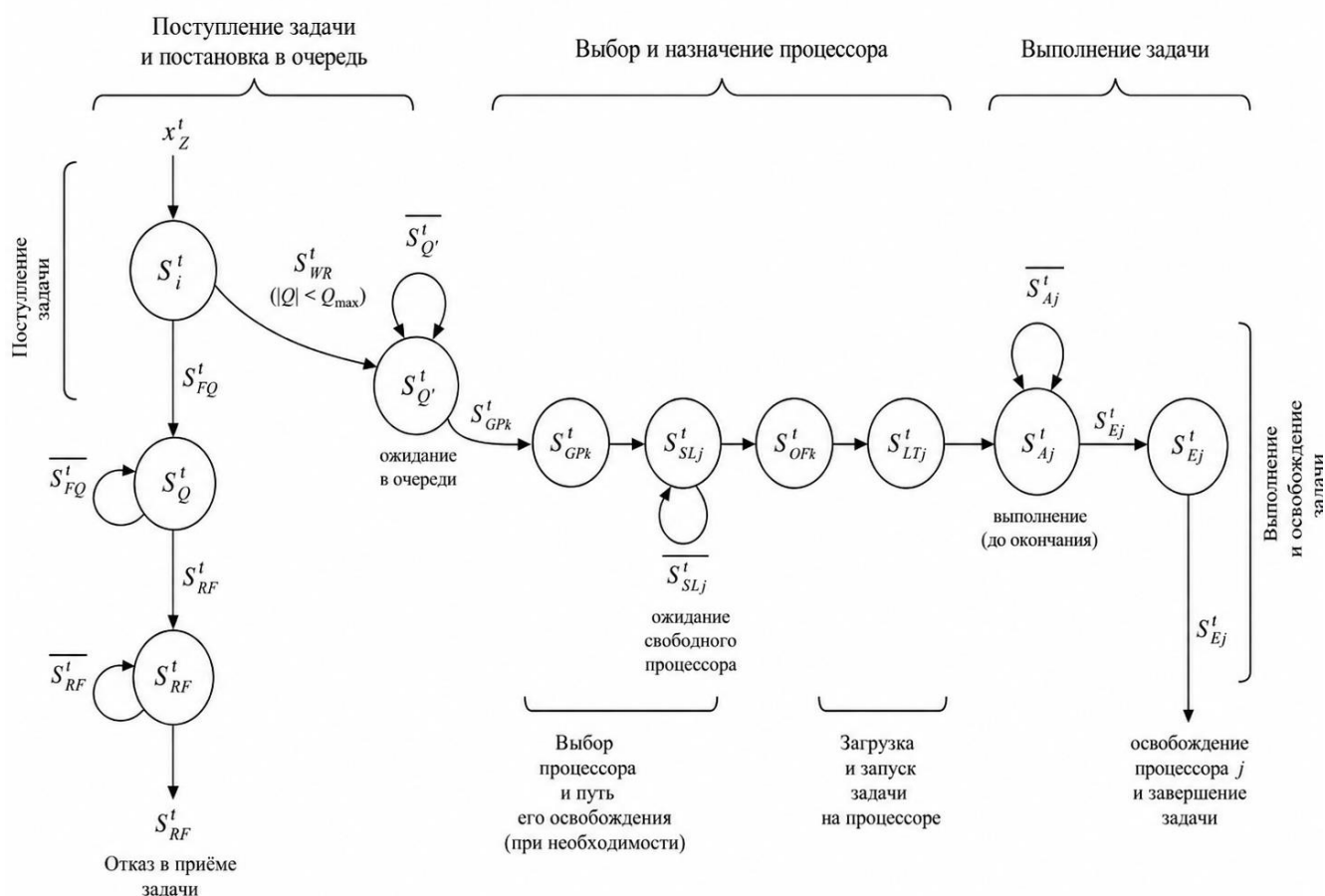


Рисунок 12 – Граф функционирования планировщика с глобальной очередью

После поступления новой заявки в систему блок генерации формирует событие появления задачи. Для данной задачи задаются идентификатор и приоритет, после чего выполняется проверка состояния глобальной очереди. Если

очередь не заполнена, заявка помещается в нее и становится доступной для дальнейшего выбора диспетчером. Данное событие можно описать формулой:

После поступления новой задачи формируется событие S_i^t . Если в глобальной очереди имеется свободное место, возникает событие записи задачи в очередь S_{WR}^t . Если свободного места нет, возникает событие отказа S_{RF}^t . При наличии задачи в очереди и свободного процессора формируются события выбора процессора S_{SLj}^p , удаления задачи из очереди S_{OFk}^t и загрузки задачи в процессор S_{LTj}^p . После загрузки процессор переходит к выполнению задачи, что описывается событием S_{Aj}^p . При завершении обслуживания возникает событие S_{Ej}^p .

Система канонических уравнений (4) НДА для модели планировщика с глобальной очередью имеет следующий вид:

$$\begin{aligned}
 S_{CR}^{t(t+1)} &= S_i^{t(t)}; \\
 S_{WR}^{t(t+1)} &= S_i^{t(t)} \cdot S_{FQ}^{t(t)}; \\
 S_{RF}^{t(t+1)} &= S_i^{t(t)} \cdot \neg S_{FQ}^{t(t)}; \\
 S_Q^{t(t+1)} &= S_{WR}^{t(t)} \vee S_Q^{t(t)} \cdot \neg (S_{OF0}^{t(t)} \vee S_{OF1}^{t(t)} \vee S_{OF2}^{t(t)} \vee S_{OF3}^{t(t)}); \\
 S_{GPK}^{t(t+1)} &= S_Q^{t(t)} \cdot x_{top_{k(t)}}; \\
 S_{SLj}^{p(t+1)} &= S_{GPK}^{t(t)} \cdot \neg S_{Bj}^{p(t)} \cdot x_{sel_{j(t)}}; \\
 S_{OFk}^{t(t+1)} &= S_{GPK}^{t(t)} \cdot S_{SLj}^{p(t)}; \\
 S_{LTj}^{p(t+1)} &= S_{SLj}^{p(t)} \cdot \neg S_{Bj}^{p(t)}; \\
 S_{Aj}^{p(t+1)} &= S_{LTj}^{p(t)} \vee S_{Aj}^{p(t)} \cdot \neg S_{Ej}^{p(t)}; \\
 S_{Ej}^{p(t+1)} &= S_{Aj}^{p(t)} \cdot x_{done_{j(t)}}.
 \end{aligned} \tag{4}$$

Обозначения частных событий и условий переходов, используемых в системе канонических уравнений НДА, приведены в таблице 1.

Первое уравнение системы (4) описывает фиксацию факта создания задачи при её поступлении в систему. Второе уравнение задаёт механизм записи задачи в глобальную очередь при наличии свободного места. Третье уравнение описывает

возникновение отказа в случае заполнения глобальной очереди. Четвёртое уравнение отражает сохранение состояния нахождения задачи в очереди до момента её удаления. Пятое уравнение определяет готовность задачи к передаче процессору. Шестое уравнение описывает выбор свободного процессора для обслуживания задачи. Седьмое уравнение задаёт удаление выбранной задачи из глобальной очереди после назначения процессора. Восьмое уравнение описывает загрузку задачи в выбранный процессор. Девятое уравнение отражает процесс выполнения задачи процессором до момента завершения обслуживания. Десятое уравнение задаёт возникновение события завершения выполнения задачи.

Представленная система канонических уравнений НДА описывает основные механизмы функционирования планировщика с глобальной очередью: создание задачи, постановку в очередь, отказ при заполнении очереди, выбор задачи, назначение процессора, удаление задачи из очереди, загрузку в процессор, выполнение и завершение обслуживания.

2.3 Формализация взаимодействия блоков планировщика с локальными очередями

В модели планировщика с локальными очередями каждая процессорная ветвь имеет собственную очередь задач. Поступающая задача сначала передаётся в блок выбора локальной очереди, после чего размещается в одной из очередей в зависимости от их текущей загрузки и наличия свободного места. В дальнейшем задача обслуживается процессором, связанным с выбранной локальной очередью.

В отличие от модели с глобальной очередью, где все задачи помещаются в общую очередь, в локальной модели дополнительно появляется механизм выбора очереди. Он позволяет распределять поступающие задачи между несколькими локальными очередями и тем самым учитывать текущую загрузку процессорных узлов. Основными блоками модели являются генератор задач, блок выбора локальной очереди, блок назначения задачи в очередь, локальные очереди, процессорные блоки и блок сбора статистики.

Обобщённая последовательность обработки задачи в модели с локальными очередями включает поступление новой задачи, анализ состояния локальных очередей, выбор очереди с минимальной загрузкой, запись задачи в выбранную очередь или отказ при заполнении всех очередей, готовность задачи к обслуживанию, загрузку задачи в соответствующий процессор, выполнение задачи и фиксацию завершения обслуживания.

Для формального описания данного процесса вводятся частные события и условия переходов НДА, представленные в таблице 2.

Таблица 2 – Обозначения частных событий и условий переходов НДА для модели планировщика с локальными очередями

Обозначение	Описание	Сигналы в модели
S_i^t	Ожидание поступления новой задачи	S_task_arrived
$S_U^{\{Qj\}}$	Состояние j-й локальной очереди	V_use_j
S_{FQj}^t	Наличие свободного места в j-й локальной очереди	$\neg S_prior_full_j$
S_{FQ}^t	Заполнение всех локальных очередей	S_all_queues_full
$S_{SL}^{\{Qj\}}$	Выбор j-й локальной очереди для размещения задачи	S_ocher_sel_j, V_selected_ocher
S_{WRj}^t	Запись задачи в j-ю локальную очередь	S_wrreq_j
S_{RF}^t	Отказ в постановке задачи в очередь	S_refuse, S_task_refused
S_{Qj}^t	Нахождение задачи в j-й локальной очереди	$V_use_j > 0$, $\neg S_prior_empty_j$
S_{GPj}^t	Готовность задачи из j-й локальной очереди к обслуживанию	S_top_valid_j
S_{OFj}^t	Удаление выбранной задачи из j-й локальной очереди	S_pop_top_j
S_{LTj}^p	Загрузка задачи в процессор с номером j	S_load_cpu_j, S_load_task
S_{Bj}^p	Занятость процессора с номером j	S_cpu_busy_j

Обозначение	Описание	Сигналы в модели
S_{Aj}^p	Выполнение задачи процессором с номером j	S_cpu_running_j
S_{Ej}^p	Завершение выполнения задачи процессором с номером j	S_task_done_j
$x_{\{min,j\}}(t)$	Условие минимальной загрузки j-й локальной очереди среди незаполненных очередей	$V_use_j = \min(V_use_1, V_use_2, V_use_3, V_use_4)$
$x_{selQj(t)}$	Условие выбора j-й локальной очереди блоком распределения задач	$V_selected_ocher = j$
$x_{topj(t)}$	Условие наличия допустимой задачи на выходе j-й локальной очереди	S_top_valid_j
$x_{donej(t)}$	Условие окончания времени обработки задачи процессором с номером j	S_task_done_j

В таблице 2 индекс j используется для обозначения номера локальной очереди и соответствующего ей процессора. Событие $S_U^{\{Qj\}}$ характеризует текущее состояние j-й локальной очереди, а условие $x_{\{min,j\}}(t)$ задаёт выбор очереди с минимальным числом занятых мест среди тех очередей, которые не заполнены.

На основании выделенных частных событий строится граф функционирования планировщика с локальными очередями, представленный на рисунке 13. Он отражает основные переходы между событиями: поступление задачи, анализ состояния локальных очередей, выбор очереди, запись задачи или отказ, ожидание задачи в локальной очереди, загрузку в соответствующий процессор, выполнение и завершение обслуживания.

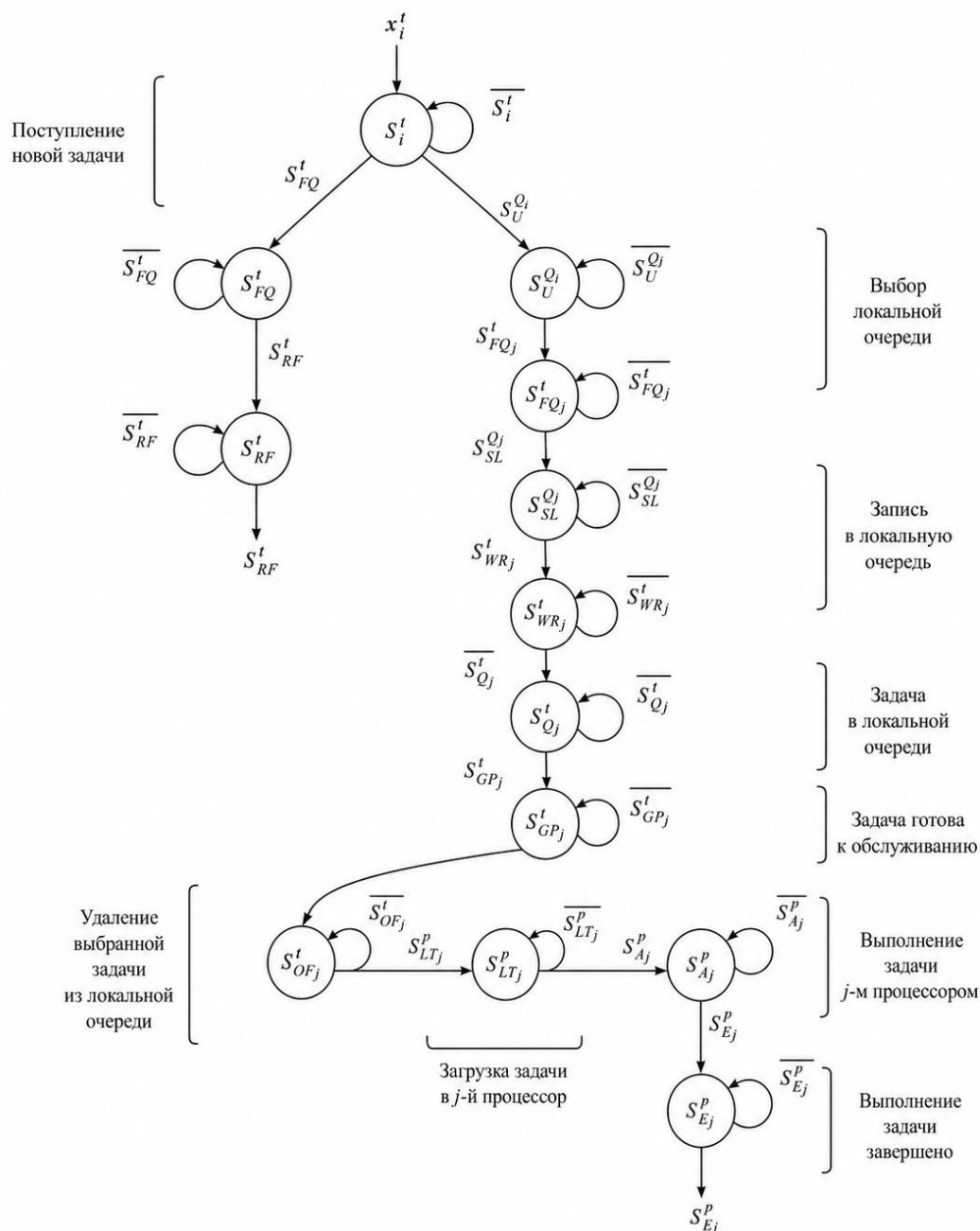


Рисунок 13 – Граф функционирования планировщика с локальными очередями

После поступления новой задачи формируется событие S_i^t . Далее анализируются состояния локальных очередей $S_U^{\{Qj\}}$. Если хотя бы одна локальная очередь имеет свободное место, выбирается очередь с минимальной загрузкой, что описывается событием $S_{SL}^{\{Qj\}}$. После выбора очереди формируется событие записи задачи S_{WRj}^t . Если все локальные очереди заполнены, возникает событие отказа

S_{RF}^t . При наличии задачи в j -й локальной очереди и свободном j -м процессоре выполняется загрузка задачи в процессор S_{LTj}^p . После загрузки процессор переходит к выполнению задачи S_{Aj}^p , а при завершении обработки формируется событие S_{Ej}^p .

Система канонических уравнений (5) НДА для модели планировщика с локальными очередями имеет следующий вид:

$$\begin{aligned}
 S_{SL}^{\{Qj\}(t+1)} &= S_i^{t(t)} \cdot S_{FQj}^{t(t)} \cdot x_{\{min,j\}(t)}; \\
 WRj^{t(t+1)} &= S_{SL}^{\{Qj\}(t)} \cdot x_{sel_{Qj}(t)}; \\
 S_{RF}^{t(t+1)} &= S_i^{t(t)} \cdot S_{FQ}^{t(t)}; \\
 S_{Qj}^{t(t+1)} &= S_{WRj}^{t(t)} \vee S_{Qj}^{t(t)} \cdot \neg S_{OFj}^{t(t)}; \\
 S_{GPj}^{t(t+1)} &= S_{Qj}^{t(t)} \cdot x_{top_{j(t)}}; \\
 S_{OFj}^{t(t+1)} &= S_{GPj}^{t(t)} \cdot \neg S_{Bj}^{p(t)}; \\
 S_{LTj}^{p(t+1)} &= S_{GPj}^{t(t)} \cdot \neg S_{Bj}^{p(t)}; \\
 S_{Aj}^{p(t+1)} &= S_{LTj}^{p(t)} \vee S_{Aj}^{p(t)} \cdot \neg S_{Ej}^{p(t)}; \\
 S_{Ej}^{p(t+1)} &= S_{Aj}^{p(t)} \cdot x_{done_{j(t)}}.
 \end{aligned} \tag{5}$$

Обозначения частных событий и условий переходов, используемых в системе канонических уравнений НДА, приведены в таблице 2.

Первое уравнение системы (5) описывает выбор j -й локальной очереди для размещения поступившей задачи при наличии свободного места и минимальной загрузке данной очереди. Второе уравнение задаёт запись задачи в выбранную локальную очередь. Третье уравнение описывает возникновение отказа при заполнении всех локальных очередей. Четвёртое уравнение отражает сохранение состояния нахождения задачи в j -й локальной очереди до момента её удаления. Пятое уравнение определяет готовность задачи к обслуживанию соответствующим процессором. Шестое уравнение задаёт удаление выбранной задачи из локальной очереди при наличии свободного процессора. Седьмое уравнение описывает

загрузку задачи в процессор с номером j . Восьмое уравнение отражает выполнение задачи процессором до момента завершения обслуживания. Девятое уравнение задаёт возникновение события завершения выполнения задачи.

Представленная система канонических уравнений НДА описывает основные механизмы функционирования планировщика с локальными очередями: выбор локальной очереди, запись задачи, отказ при заполнении всех очередей, ожидание задачи в очереди, загрузку в соответствующий процессор, выполнение и завершение обслуживания.

2.4 Формализация бесприоритетного режима обслуживания

Бесприоритетный режим обслуживания соответствует дисциплине FIFO, при которой задачи выбираются из очереди в порядке их поступления. В данном режиме значение приоритета может сохраняться как информационный параметр задачи и использоваться при сборе статистики, однако оно не влияет на порядок выбора задачи для обслуживания.

Для модели с глобальной очередью это означает, что из общей очереди выбирается задача, поступившая раньше остальных доступных задач. Для модели с локальными очередями аналогичный принцип применяется внутри каждой локальной очереди: процессору с номером j передаётся первая задача из соответствующей j -й очереди.

В режиме FIFO отсутствует механизм вытеснения выполняемой задачи. После загрузки задачи в процессор она продолжает выполняться до окончания времени обработки. Новые поступающие задачи помещаются в очередь и ожидают освобождения процессора.

Для описания режима FIFO вводятся следующие условия выбора:

$x_{\{FIFO,k\}}(t)$ – условие выбора из глобальной очереди задачи, поступившей раньше остальных доступных задач;

$x_{\{FIFO,j\}}(t)$ – условие выбора первой задачи из j -й локальной очереди.

Система канонических уравнений (6) НДА для беспriorитетного режима обслуживания имеет следующий вид:

$$\begin{aligned}
 S_{\{GPK\}}^{t(t+1)} &= S_Q^{t(t)} \cdot x_{\{FIFO,k\}}(t); \\
 S_{\{GPj\}}^{t(t+1)} &= S_{\{Qj\}}^{t(t)} \cdot x_{\{FIFO,j\}}(t); \\
 S_{\{LTj\}}^{p(t+1)} &= S_{\{GPj\}}^{t(t)} \cdot \neg S_{\{Bj\}}^{p(t)}; \\
 S_{\{Aj\}}^{p(t+1)} &= S_{\{LTj\}}^{p(t)} \vee S_{\{Aj\}}^{p(t)} \cdot \neg S_{\{Ej\}}^{p(t)}; \\
 S_{\{Ej\}}^{p(t+1)} &= S_{\{Aj\}}^{p(t)} \cdot x_{\{done,j\}}(t).
 \end{aligned} \tag{6}$$

Обозначения частных событий, используемых в системе канонических уравнений НДА, приведены в таблицах 1 и 2.

Первое уравнение системы (6) описывает выбор задачи из глобальной очереди в порядке поступления. Второе уравнение задаёт выбор первой задачи из j-й локальной очереди. Третье уравнение описывает загрузку задачи в свободный процессор. Четвёртое уравнение отражает выполнение задачи без вытеснения до момента завершения обслуживания. Пятое уравнение задаёт возникновение события завершения выполнения задачи.

Таким образом, в беспriorитетном режиме обслуживания порядок обработки определяется только временем поступления задач в очередь. Приоритет задачи не участвует в формировании условий выбора, поэтому данный режим используется как базовый вариант планирования для последующего сравнения с режимами относительных и абсолютных приоритетов.

Представленная система канонических уравнений НДА описывает основные механизмы функционирования планировщика с локальными очередями: выбор локальной очереди, запись задачи, отказ при заполнении всех очередей, ожидание задачи в очереди, загрузку в соответствующий процессор, выполнение и завершение обслуживания.

2.5 Формализация режима обслуживания с относительными приоритетами

Режим обслуживания с относительными приоритетами предполагает, что при выборе задачи из очереди учитывается значение её приоритета. В отличие от беспriorитетного режима, задача выбирается не только по порядку поступления, а по максимальному значению приоритета среди задач, находящихся в очереди.

Для модели с глобальной очередью это означает, что блок выбора анализирует задачи, находящиеся в общей очереди, и передаёт на обслуживание задачу с наибольшим приоритетом. Для модели с локальными очередями выбор задачи с максимальным приоритетом выполняется внутри каждой локальной очереди. При этом уже выполняемая процессором задача не прерывается. Если в очередь поступает задача с более высоким приоритетом, она ожидает освобождения процессора и будет выбрана раньше задач с меньшим приоритетом.

Таким образом, относительный приоритет влияет только на порядок выбора задачи из очереди, но не изменяет процесс выполнения уже загруженной задачи. После загрузки в процессор задача выполняется до завершения времени обработки.

Для описания режима относительных приоритетов вводятся следующие условия выбора:

$x_{\{PR,k\}(t)}$ – условие выбора задачи с максимальным приоритетом из глобальной очереди;

$x_{\{PR,j\}(t)}$ – условие выбора задачи с максимальным приоритетом из j -й локальной очереди.

Система канонических уравнений (7) НДА для режима обслуживания с относительными приоритетами имеет следующий вид:

$$\begin{aligned} S_{GPK}^{t(t+1)} &= S_Q^{t(t)} \cdot x_{\{PR,k\}(t)}; \\ S_{GPj}^{t(t+1)} &= S_{Qj}^{t(t)} \cdot x_{\{PR,j\}(t)}; \\ S_{LTj}^{p(t+1)} &= S_{GPj}^{t(t)} \cdot \neg S_{Bj}^{p(t)}; \end{aligned} \quad (7)$$

$$S_{Aj}^{p(t+1)} = S_{LTj}^{p(t)} \vee S_{Aj}^{p(t)} \cdot \neg S_{Ej}^{p(t)};$$

$$S_{Ej}^{p(t+1)} = S_{Aj}^{p(t)} \cdot x_{\{done,j\}(t)}.$$

Обозначения частных событий, используемых в системе канонических уравнений НДА, приведены в таблицах 1 и 2.

Первое уравнение системы (7) описывает выбор задачи с максимальным приоритетом из глобальной очереди. Второе уравнение задаёт выбор задачи с максимальным приоритетом из j -й локальной очереди. Третье уравнение описывает загрузку выбранной задачи в свободный процессор. Четвёртое уравнение отражает выполнение задачи без вытеснения до момента завершения обслуживания. Пятое уравнение задаёт возникновение события завершения выполнения задачи.

Режим относительных приоритетов отличается от беспriorитетного режима только правилом выбора задачи из очереди. В FIFO-режиме выбирается задача, поступившая раньше остальных, а в режиме относительных приоритетов выбирается задача с наибольшим значением приоритета. При этом механизм постановки задачи в очередь, загрузки в процессор и завершения обслуживания остаётся неизменным.

2.6 Формализация режима обслуживания с абсолютными приоритетами

Режим обслуживания с абсолютными приоритетами отличается от беспriorитетного режима и режима относительных приоритетов наличием механизма вытеснения. В данном режиме поступившая задача с более высоким приоритетом может прервать выполнение задачи с меньшим приоритетом, уже находящейся на процессоре. При этом параметры вытесненной задачи сохраняются, после чего она возвращается в очередь для последующего обслуживания.

В модели с глобальной очередью более приоритетная задача может быть назначена одному из занятых процессоров, если выполняемая на нём задача имеет

меньший приоритет. В модели с локальными очередями аналогичная логика применяется внутри соответствующей процессорной ветви. При возникновении вытеснения текущая задача снимается с выполнения, её оставшееся время обработки и служебные параметры сохраняются, а более приоритетная задача загружается в процессор.

Для формального описания режима абсолютных приоритетов вводятся дополнительные частные события и условия переходов НДА, представленные в таблице 3.

Таблица 3 – Обозначения дополнительных частных событий и условий переходов НДА для режима абсолютных приоритетов

Обозначение	Описание	Сигналы в модели
S_{INTj}^p	Прерывание выполнения задачи процессором с номером j	$S_interrupt_j$
S_{PRMj}^p	Сохранение параметров вытесненной задачи	$S_preempted_valid_j$, $V_preempted_task_id_j$, $V_preempted_prior_j$, $V_preempted_rem_j$
S_{RQj}^t	Возврат вытесненной задачи в очередь	S_wrreq_j , данные $V_preempted_*$
$x_{\{high,j\}}(t)$	Условие наличия задачи с более высоким приоритетом, чем у задачи, выполняемой j -м процессором	$V_task_prior >$ $V_current_prior_j$
$x_{\{save,j\}}(t)$	Условие сохранения параметров вытесненной задачи	$S_preempted_valid_j$
$x_{\{ret,j\}}(t)$	Условие возврата вытесненной задачи в очередь	Формирование записи вытесненной задачи в очередь

В таблице 3 индекс j используется для обозначения номера процессора, на котором выполняется задача, подлежащая возможному вытеснению. Событие S_{INTj}^p соответствует факту прерывания работы процессора, событие S_{PRMj}^p – сохранению

параметров вытесненной задачи, а событие S_{RQj}^t – возврату вытесненной задачи в очередь. Условие $x_{\{high,j\}}(t)$ описывает сравнение приоритетов поступившей и выполняемой задач.

Механизм абсолютных приоритетов является дополнительной ветвью графа функционирования планировщика, представленный на рисунке 14. Он включается в том случае, если в систему поступает задача с более высоким приоритетом, чем задача, уже выполняемая одним из процессоров.

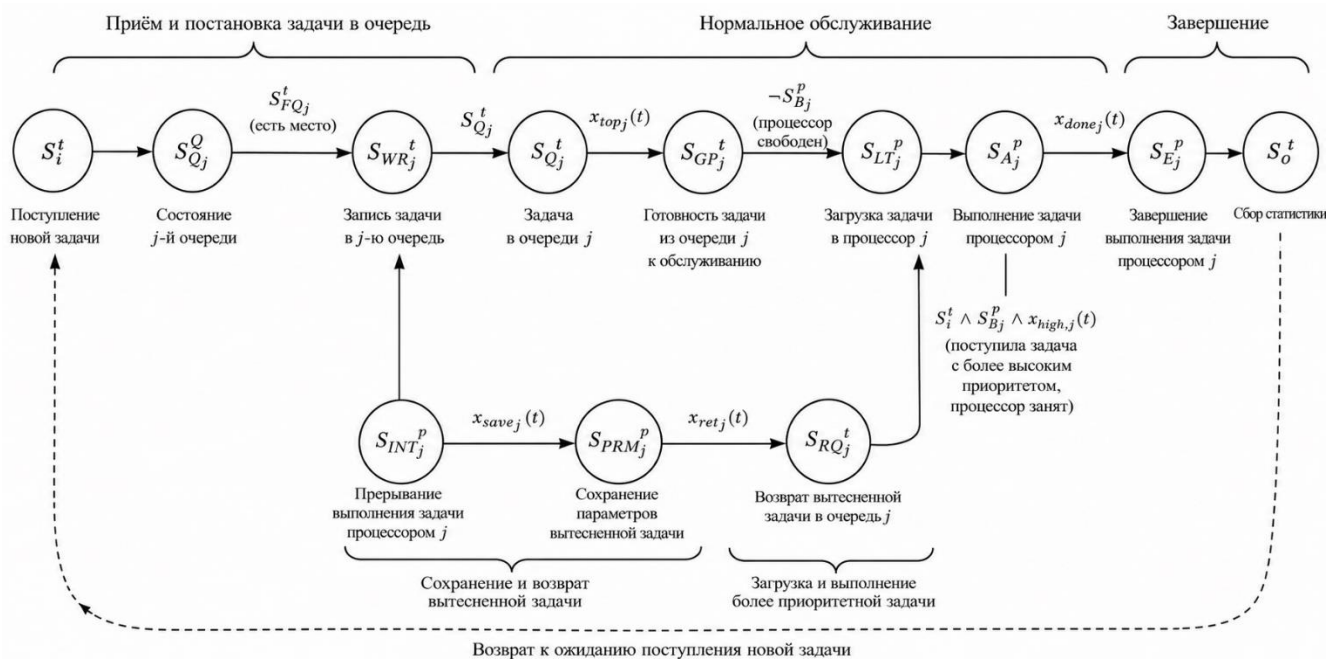


Рисунок 14 – Фрагмент графа функционирования планировщика в режиме абсолютных приоритетов

При поступлении более приоритетной задачи и занятости j -го процессора проверяется условие $x_{\{high,j\}}(t)$. Если выполняемая процессором задача имеет меньший приоритет, возникает событие прерывания S_{INTj}^p . После этого фиксируются параметры вытесненной задачи: номер задачи, её приоритет, оставшееся время обработки и накопленные временные характеристики. Данный этап описывается событием S_{PRMj}^p . Затем более приоритетная задача загружается в

процессор, а вытесненная задача возвращается в очередь, что описывается событием S_{RQj}^t .

Система канонических уравнений (8) НДА для режима обслуживания с абсолютными приоритетами имеет следующий вид:

$$\begin{aligned}
 S_{INTj}^{p(t+1)} &= S_i^{t(t)} \cdot S_{Bj}^{p(t)} \cdot x_{\{high,j\}(t)}; \\
 S_{PRMj}^{p(t+1)} &= S_{INTj}^{p(t)} \cdot x_{\{save,j\}(t)}; \\
 S_{RQj}^{t(t+1)} &= S_{PRMj}^{p(t)} \cdot x_{\{ret,j\}(t)}; \\
 S_{LTj}^{p(t+1)} &= S_i^{t(t)} \cdot S_{INTj}^{p(t)}; \\
 S_{Aj}^{p(t+1)} &= S_{LTj}^{p(t)} \vee S_{Aj}^{p(t)} \cdot \neg S_{Ej}^{p(t)}; \\
 S_{Ej}^{p(t+1)} &= S_{Aj}^{p(t)} \cdot x_{\{done,j\}(t)};
 \end{aligned} \tag{8}$$

Обозначения частных событий и условий переходов, используемых в системе канонических уравнений НДА, приведены в таблицах 1–3.

Первое уравнение системы (8) описывает прерывание выполняемой задачи при поступлении новой задачи с более высоким приоритетом. Второе уравнение задаёт сохранение параметров вытесненной задачи. Третье уравнение определяет возврат вытесненной задачи в очередь. Четвёртое уравнение описывает загрузку более приоритетной задачи в процессор после выполнения прерывания. Пятое уравнение отражает дальнейшее выполнение загруженной задачи. Шестое уравнение задаёт завершение выполнения задачи.

Таким образом, режим абсолютных приоритетов дополняет ранее описанные механизмы постановки задачи в очередь, выбора и загрузки в процессор ветвью вытеснения. В отличие от режима относительных приоритетов, более приоритетная задача не ожидает освобождения процессора, а может прервать выполнение менее приоритетной задачи. За счёт сохранения параметров вытесненной задачи обеспечивается возможность её последующего возврата в очередь и продолжения обслуживания.

3 Аппаратная реализация алгоритмов приоритетного планирования

3.1 Проблемы управления ресурсами и процессами в многопроцессорных системах

В многопроцессорных вычислительных системах управление процессами и ресурсами является одной из основных функций операционной системы. К таким функциям относятся планирование и диспетчеризация задач, управление памятью, обработка прерываний, синхронизация процессов и организация межпроцессного взаимодействия. При традиционной программной реализации эти операции выполняются ядром операционной системы и требуют дополнительных временных затрат, поскольку вызываются многократно в процессе работы системы. В результате часть процессорного времени расходуется не на выполнение прикладных задач, а на выполнение служебных функций управления.

Для систем реального времени указанные временные затраты имеют особое значение, так как такие системы должны обеспечивать выполнение задач в пределах заданных временных ограничений. Даже небольшие задержки, возникающие при работе планировщика, диспетчера или механизмов синхронизации, могут привести к увеличению времени реакции системы. Поэтому при разработке многопроцессорных систем реального времени важно не только повысить общую производительность вычислений, но и уменьшить задержки, связанные с выполнением служебных функций операционной системы.

В многопроцессорных системах задача управления процессами усложняется по сравнению с однопроцессорными системами. Если в однопроцессорной системе планировщик должен выбрать только следующую задачу для выполнения, то в многопроцессорной системе необходимо также определить процессор, на котором эта задача будет выполняться. Таким образом, задача планирования и диспетчеризации становится более сложной, так как выбор задачи, выбор процессора и согласование их взаимодействия.

Дополнительная сложность связана с тем, что несколько процессоров могут одновременно обращаться к общим структурам данных операционной системы. К таким структурам относятся очереди готовых задач, таблицы состояний процессов, общие ресурсы и средства межпроцессного взаимодействия. Одновременный доступ к ним может приводить к конфликтам, для устранения которых требуются механизмы синхронизации. Однако сами механизмы синхронизации также создают дополнительные временные затраты и могут снижать эффективность работы системы.

Одним из важных элементов управления процессами является организация очередей готовых задач. В очередях хранятся задачи, ожидающие назначения на процессор. От способа организации очередей зависит время ожидания задач, равномерность загрузки процессоров и сложность реализации планировщика. В многопроцессорных системах могут применяться различные способы организации очередей, в том числе общая очередь для всех процессоров или отдельные локальные очереди, связанные с конкретными процессорами.

Приоритетное обслуживание задач также усложняет работу планировщика. В таких системах задачи имеют разные уровни важности, поэтому планировщик должен обеспечивать преимущественное обслуживание более приоритетных задач. При этом возможны различные режимы обслуживания: с вытеснением выполняющейся задачи, без вытеснения или без учёта приоритетов. Каждый из этих режимов по-разному влияет на временные характеристики системы и сложность аппаратной реализации.

Одним из способов уменьшения временных затрат на выполнение функций управления процессами является аппаратная реализация отдельных функций операционной системы. В этом случае часть операций, выполняемых программным ядром ОС, переносится в специализированные аппаратные блоки. Такие блоки могут выполнять выбор задачи, управление очередями, выбор процессора и назначение задачи на выполнение быстрее, чем программная реализация.

Однако аппаратная реализация имеет и ограничения. В отличие от программного алгоритма, аппаратную схему сложнее изменять и адаптировать под новые требования. Поэтому при разработке аппаратных средств управления процессами необходимо заранее определить структуру планировщика, способ организации очередей и режим обслуживания задач. Это делает актуальным исследование различных вариантов аппаратных планировщиков в условиях многопроцессорной системы.

Таким образом, основные проблемы управления процессами и ресурсами в многопроцессорных системах связаны с накладными расходами программной реализации функций ОС, необходимостью выбора задачи и процессора, конфликтами при доступе к общим структурам данных, организацией очередей задач и поддержкой приоритетного обслуживания. Для снижения этих затрат далее рассматривается архитектура аппаратного ядра многопроцессорной операционной системы и аппаратные планировщики задач с различной организацией очередей.

3.2 Архитектура аппаратного ядра операционной системы

Одним из способов уменьшения временных затрат, связанных с выполнением функций операционной системы, является перенос части этих функций в специализированные аппаратные блоки. Такой подход позволяет ускорить выполнение операций управления процессами и ресурсами, которые в программной реализации требуют дополнительного процессорного времени. В многопроцессорных системах аппаратная поддержка особенно важна, поскольку управление задачами связано не только с выбором очередной задачи, но и с выбором процессора, синхронизацией доступа к общим структурам данных и контролем состояния вычислительных узлов.

Аппаратное ядро операционной системы можно рассматривать как специализированную аппаратную подсистему, предназначенную для выполнения наиболее часто используемых функций управления. Оно не заменяет операционную систему полностью, а реализует те функции, которые являются

критичными по времени: планирование, диспетчеризацию, управление очередями, обработку запросов к ресурсам и синхронизацию взаимодействующих процессов.

Аппаратное ядро может быть подключено к многопроцессорной системе различными способами. Наиболее простой вариант предполагает использование общей системной шины, через которую осуществляется обмен между процессорами, оперативной памятью и аппаратным ядром ОС. В этом случае аппаратное ядро выступает как одно из устройств, подключённых к общей шине, и получает доступ к управляющей информации через тот же канал, что и остальные компоненты системы. Такой подход удобен простотой интеграции, поскольку не требует выделения дополнительных каналов связи. Однако его недостатком является возможная конкуренция за шину: обращения процессоров к памяти и обращения к аппаратному ядру могут выполняться через один и тот же канал. Подключение аппаратного ядра ОС через общую системную шину представлено на рисунке 15.



Рисунок 15 – Подключение аппаратного ядра ОС через общую системную шину

Для снижения нагрузки на общую системную шину может использоваться вариант с выделенной шиной управления. В такой архитектуре основной обмен данными между процессорами и памятью выполняется по системной шине, а обращения к аппаратному ядру ОС передаются по отдельному каналу. Это

позволяет разделить потоки данных и управляющих запросов: системная шина используется для выполнения прикладных операций, а выделенная шина — для взаимодействия с аппаратным ядром. Преимуществом такого решения является снижение конкуренции за основной канал обмена и более предсказуемая передача управляющей информации. Недостатком является усложнение аппаратной структуры, так как требуется дополнительный интерфейс и отдельные линии связи. Подключение аппаратного ядра ОС через выделенную шину управления представлено на рисунке 18.



Рисунок 16 – Подключение аппаратного ядра ОС через выделенную шину управления

При подключении аппаратного ядра к общей шине его внутренняя структура должна обеспечивать приём запросов от процессоров, обмен с памятью и передачу управляющих сигналов между внутренними блоками ядра. В такой структуре интерфейс ядра является связующим элементом между общей системной шиной и внутренними блоками аппаратного ядра. Через этот интерфейс поступают запросы на выполнение функций управления процессами и ресурсами. Далее они передаются в соответствующие функциональные блоки: блок управления

процессами или блок управления ресурсами. Такой вариант организации позволяет централизовать взаимодействие аппаратного ядра с остальными компонентами системы, но его эффективность зависит от доступности общей шины. Структурная организация аппаратного ядра ОС при подключении к общей шине показана на рисунке 17.

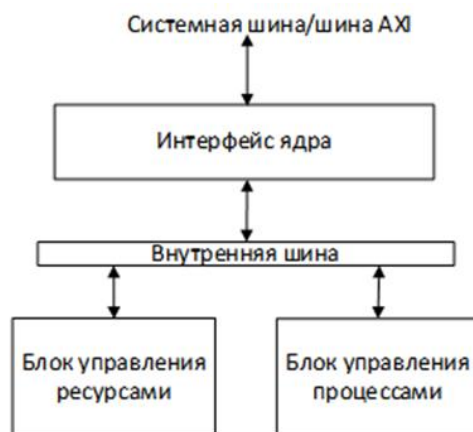


Рисунок 17 – Структурная организация аппаратного ядра ОС при подключении к общей шин

Другой способ построения аппаратного ядра основан на использовании непосредственных связей с процессорами. В этом случае каждый процессор может иметь отдельные линии для передачи запросов, признаков занятости, сигналов прерывания и другой управляющей информации. Такая организация уменьшает зависимость обмена управляющими сигналами от общей шины и может обеспечить более быструю реакцию аппаратного ядра на изменение состояния процессоров. Это особенно важно для планировщика задач, которому необходимо оперативно получать информацию о занятости процессоров и передавать им команды на загрузку задач.

Вместе с тем при увеличении числа процессоров растёт количество соединений, что усложняет аппаратную реализацию. Структурная организация аппаратного ядра ОС при использовании прямых связей с процессорами представлена на рисунке 18.



Рисунок 18 – Структурная организация аппаратного ядра ОС при использовании прямых связей с процессорами

Вариант с общей памятью предполагает, что процессоры и аппаратное ядро используют единую область памяти для хранения управляющей информации. В такой памяти могут размещаться состояния задач, признаки занятости процессоров, сведения о ресурсах, очереди ожидания и другие данные, необходимые для принятия управляющих решений. Преимущество данного подхода заключается в том, что все участники системы работают с единой информационной областью. Однако при этом возникает необходимость согласования доступа к общей памяти, так как несколько процессоров и аппаратное ядро могут обращаться к одним и тем же данным. Поэтому такая структура требует механизмов арбитража и синхронизации доступа. Структурная организация аппаратного ядра ОС с использованием общей памяти показана на рисунке 19.



Рисунок 19 – Структурная организация аппаратного ядра ОС с использованием

В составе аппаратного ядра ОС одним из основных элементов является блок управления процессами. Он отвечает за выполнение функций, связанных с жизненным циклом задач: поступлением задач в систему, хранением их в очередях готовности, выбором задачи для выполнения, определением доступного процессора и передачей задачи на выполнение. В многопроцессорной системе этот блок должен учитывать состояние нескольких процессоров и обеспечивать согласованную диспетчеризацию задач. Если используется приоритетное обслуживание, блок управления процессами также должен учитывать приоритеты задач, а при абсолютном приоритете — поддерживать механизм вытеснения менее приоритетных задач. Функциональный состав блока управления процессами аппаратного ядра ОС показан на рисунке 20.

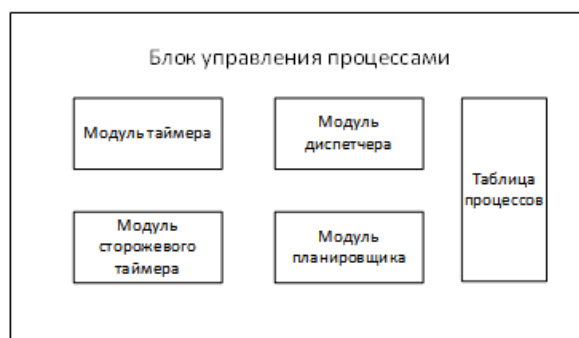


Рисунок 20 – Функциональный состав блока управления процессами аппаратного ядра ОС

Вторым важным элементом аппаратного ядра является блок управления ресурсами. Его назначение связано не с выбором задач для выполнения, а с организацией доступа процессов к разделяемым ресурсам системы. К таким ресурсам могут относиться области памяти, устройства ввода-вывода, каналы обмена и другие общие объекты. Блок управления ресурсами должен учитывать занятость ресурсов, обрабатывать запросы на их получение, формировать очереди ожидания и разрешать конфликты при одновременных обращениях. Таким образом, если блок управления процессами определяет порядок выполнения задач, то блок управления ресурсами определяет порядок доступа к общим ресурсам.

Функциональный состав блока управления ресурсами аппаратного ядра ОС приведён на рисунке 21.

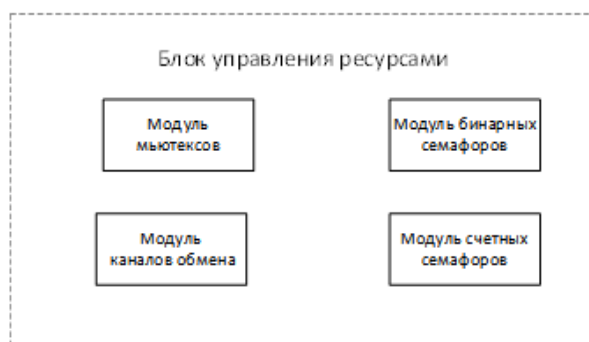


Рисунок 21 – Функциональный состав блока управления ресурсами аппаратного ядра ОС

В рамках данной работы основное внимание уделяется не всему аппаратному ядру ОС, а его части, связанной с планированием и диспетчеризацией задач. Эту часть можно представить как аппаратный планировщик-диспетчер, который принимает сведения о поступающих задачах, хранит их в очередях, получает информацию о состоянии процессоров и формирует сигналы назначения задач на выполнение. В зависимости от выбранной архитектуры планировщик может использовать одну глобальную очередь или несколько локальных очередей. Кроме того, в нём могут быть реализованы различные режимы обслуживания: абсолютное приоритетное, относительное приоритетное или беспriorитетное. Схема подключения аппаратного планировщика-диспетчера задач к многопроцессорной системе показана на рисунке 22.



Рисунок 22 – Подключение аппаратного планировщика-диспетчера задач к многопроцессорной системе

3.3 Моделирование аппаратного планировщика с глобальной очередью

Модель аппаратного планировщика с глобальной очередью была реализована на языке VHDL в виде набора программных модулей. Ввод описания, компиляция и имитационное моделирование выполнялись в среде Xilinx 9.1. Основу модели составляют блок генерации задач, блок управления глобальной очередью, блок глобальной очереди, блок выбора процессора, блок синхронизации и назначения задач, четыре процессорных блока и блок сбора статистики.

Структурно модель планировщика с глобальной очередью состоит из следующих основных блоков:

- блока генерации задач;
- блока управления глобальной очередью;
- блока глобальной очереди;
- блока выбора процессора;
- блока синхронизации и назначения задач;
- процессорных блоков;

– блока сбора статистики.

В модели используется единая глобальная очередь, общая для всех процессоров. Все поступающие задачи сначала направляются в эту очередь, после чего при освобождении одного или нескольких процессоров выбираются из неё и передаются на обслуживание. Такой способ организации позволяет каждому свободному процессору получать задачу из общего набора ожидающих задач.

Блок генерации задач формирует входной поток задач. Каждой новой задаче присваивается идентификатор и значение приоритета. Идентификатор передаётся по шине V_task_id, приоритет – по шине V_task_prior, а факт появления новой задачи фиксируется сигналом S_task_arrived. Интервалы между поступлениями задач формируются с использованием генератора случайных чисел и параметра LAMBDA_C, задающего интенсивность входного потока. Приоритет задачи задаётся четырёхразрядным значением, что позволяет использовать шестнадцать уровней приоритета.

Блок управления глобальной очередью предназначен для обработки сигнала поступления новой задачи и контроля возможности её помещения в глобальную очередь. При поступлении задачи блок анализирует состояние очереди и формирует сигнал записи задачи либо сигнал отказа. Для этого используются сигналы S_queue_full, S_queue_empty и шина V_queue_length, содержащая текущее число задач в очереди. В модели также формируются сигналы S_new_wtreq, V_new_task_id, V_new_task_prior, S_refuse, S_task_created и V_created_prior.

Возможность записи новой задачи определяется не только текущей заполненностью глобальной очереди, но и количеством задач, удаляемых из очереди в текущем такте. Для этого учитываются сигналы S_pop_top_0–S_pop_top_3. За счёт этого при освобождении места в очереди новая задача может быть записана в неё без дополнительной задержки. В режиме абсолютных приоритетов дополнительно учитываются вытесненные задачи, возвращаемые в очередь.

Блок глобальной очереди хранит задачи, ожидающие передачи на процессоры. В VHDL-модели очередь имеет ограниченную ёмкость, задаваемую

параметром QUEUE_SIZE_C. В составе каждой записи очереди сохраняются идентификатор задачи, её приоритет, время ожидания, время пребывания в системе, оставшееся время обслуживания и признак того, что задача уже начинала выполняться. Состояние очереди характеризуется сигналами S_queue_full, S_queue_empty и V_queue_length.

На выходе глобальной очереди формируются признаки готовности верхних элементов очереди и их параметры. Для одновременного назначения задач нескольким процессорам используются сигналы S_top_valid_0–S_top_valid_3, а также соответствующие шины данных V_top_task_id_0–V_top_task_id_3, V_top_prior_0–V_top_prior_3, V_top_wait_time_0–V_top_wait_time_3, V_top_system_time_0–V_top_system_time_3, V_top_rem_time_0–V_top_rem_time_3, V_top_started_0–V_top_started_3. Удаление выбранных задач из очереди выполняется сигналами S_pop_top_0–S_pop_top_3. После удаления выбранных задач оставшиеся элементы сдвигаются к началу очереди. На каждом такте для всех задач, находящихся в очереди, увеличиваются значения времени ожидания и времени пребывания в системе.

Блок выбора процессора определяет, на какой процессор должна быть передана очередная задача из глобальной очереди. Для этого анализируются сигналы занятости процессоров S_cpu_busy_0–S_cpu_busy_3 и сигналы наличия верхних задач очереди S_top_valid_0–S_top_valid_3. При наличии нескольких свободных процессоров используется циклический принцип выбора. Он позволяет последовательно распределять задачи между процессорами и не закреплять все новые назначения за одним и тем же процессором.

Схема программы выбора процессора в планировщике с глобальной очередью представлена на рисунке 23.

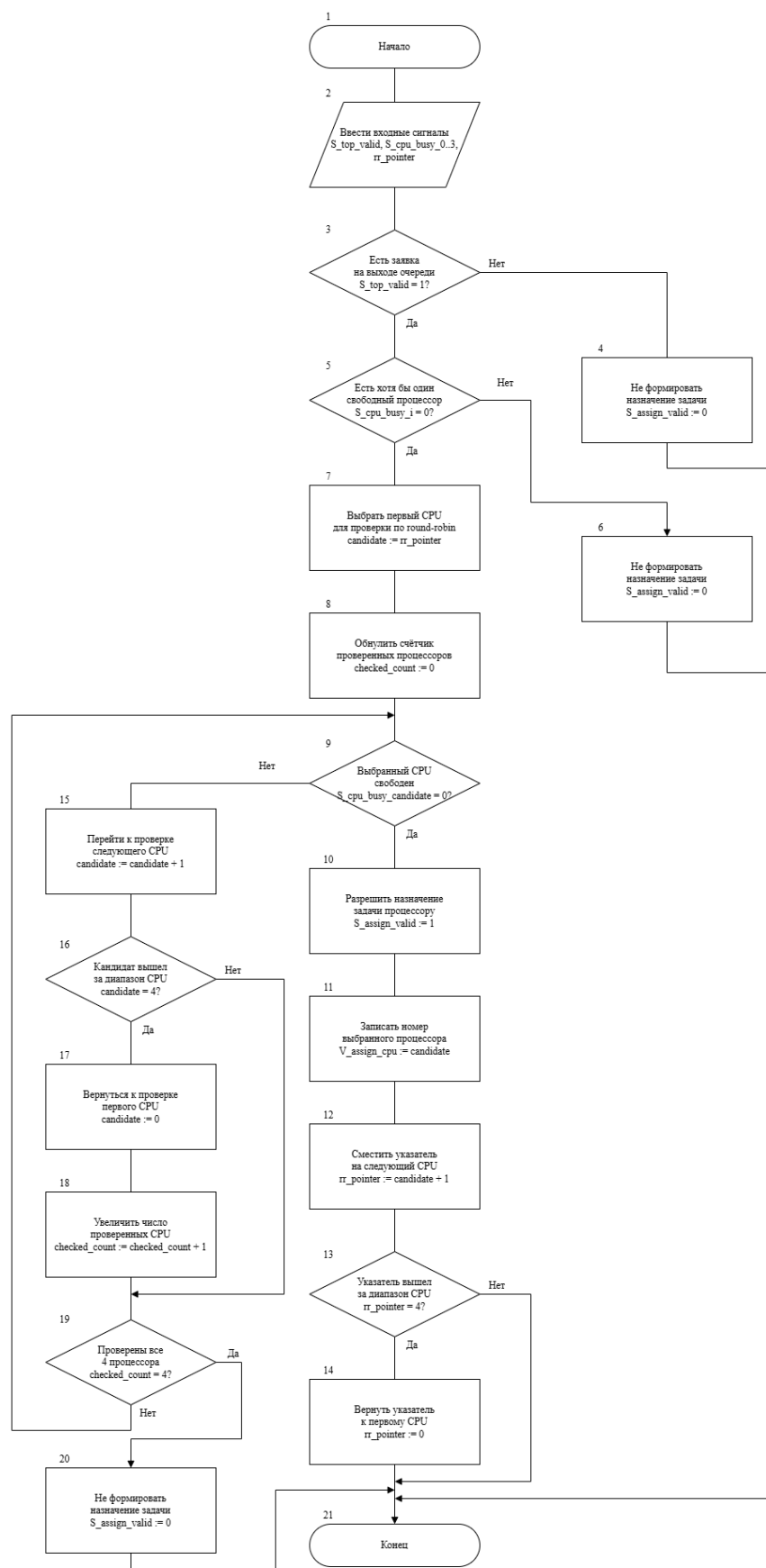


Рисунок 23 – Схема программы выбора процессора в планировщике с глобальной очередью

Блок синхронизации и назначения задач связывает глобальную очередь с процессорными блоками. Он получает от блока выбора процессора признаки назначения и номера выбранных процессоров, а от глобальной очереди – параметры задач. На основании этих данных формируются сигналы загрузки процессоров $S_load_cpu_0$ – $S_load_cpu_3$, которые подключаются к входам S_load_task соответствующих процессорных блоков. На вход процессорного блока передаются идентификатор задачи, её приоритет, накопленное время ожидания, время пребывания в системе, оставшееся время обслуживания и признак повторного запуска задачи.

Процессорный блок имитирует выполнение задачи. После получения сигнала загрузки процессор переходит из свободного состояния в состояние выполнения. Время обслуживания формируется случайным образом с использованием параметра MU_C . В процессе обслуживания уменьшается оставшееся время выполнения задачи и увеличивается время её пребывания в системе. После завершения обслуживания процессор формирует сигнал окончания задачи и снова переходит в свободное состояние. Состояние процессора характеризуется сигналами S_cpu_busy и $S_cpu_running$, а завершение обработки фиксируется сигналом S_task_done .

Схема программы выполнения задачи процессорным блоком представлена на рисунке 24.

В рамках модели с глобальной очередью реализованы три режима обслуживания задач: бесприоритетный режим, режим относительных приоритетов и режим абсолютных приоритетов.

В бесприоритетном режиме глобальная очередь работает по дисциплине FIFO. Новая задача записывается в конец очереди, а на обслуживание передаётся задача, находящаяся в начале очереди. Значение приоритета в этом режиме сохраняется вместе с задачей, но не влияет на порядок выбора. Такой вариант используется как базовый для сравнения с приоритетными дисциплинами обслуживания.

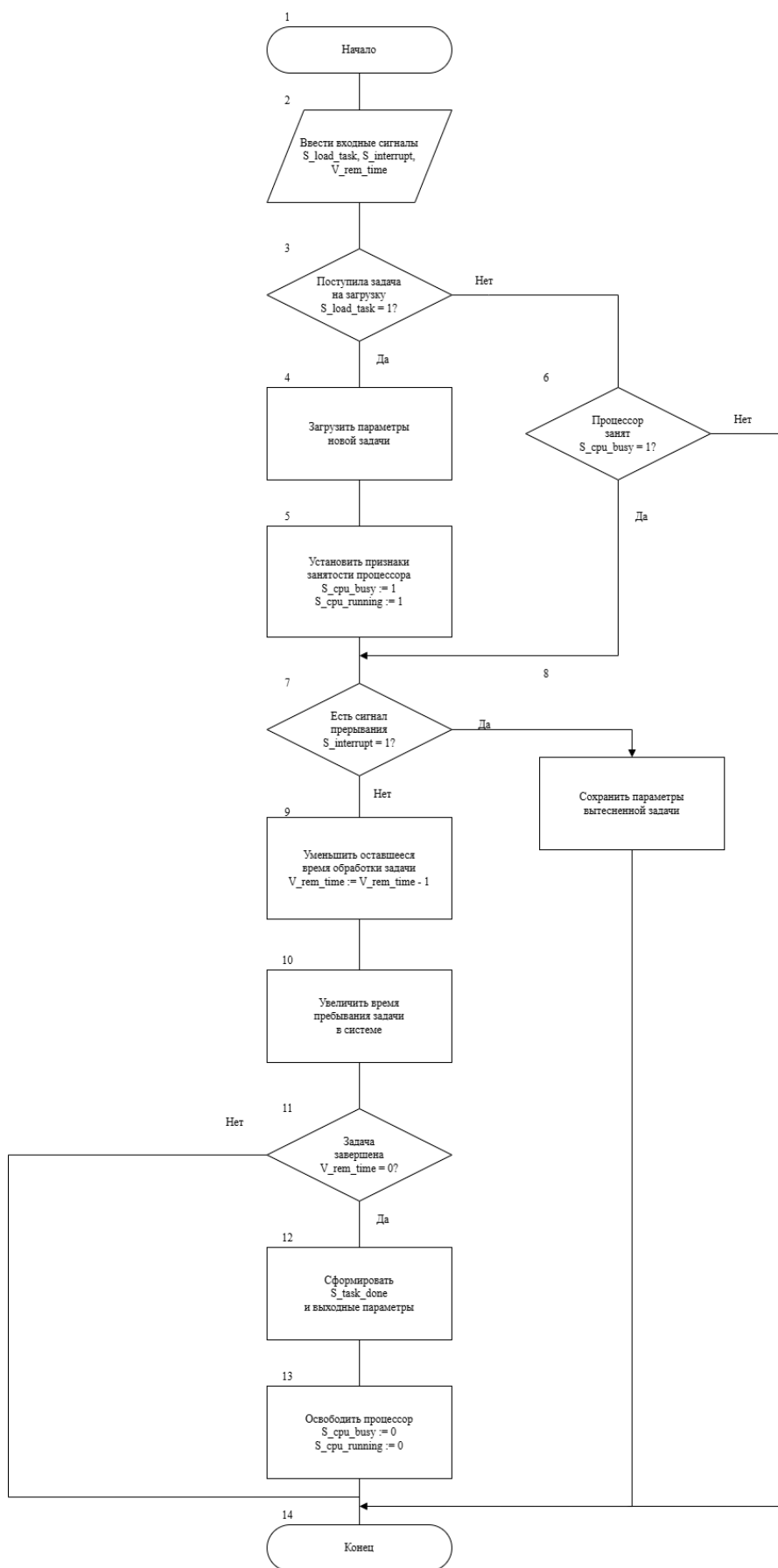


Рисунок 24 – Схема программы выполнения задачи процессорным блоком

В режиме относительных приоритетов задачи в глобальной очереди упорядочиваются по значению приоритета. На обслуживание в первую очередь передаются задачи с более высоким приоритетом. В VHDL-модели более высокий приоритет соответствует меньшему числовому значению поля V_task_prior. Если несколько задач имеют одинаковый приоритет, сохраняется порядок их поступления. В этом режиме приоритет влияет только на выбор следующей задачи из очереди. Уже выполняемая задача не прерывается и занимает процессор до завершения обслуживания.

Схема программы обслуживания задач в режиме относительных приоритетов представлена на рисунке 25.

В режиме абсолютных приоритетов дополнительно реализован механизм вытеснения. Если в систему поступает задача с более высоким приоритетом, а свободных процессоров нет, выполняется сравнение её приоритета с приоритетами задач, находящихся на процессорах. В VHDL-модели более высокий приоритет также соответствует меньшему числовому значению поля приоритета. При наличии менее приоритетной выполняемой задачи формируется сигнал прерывания S_interrupt.

Параметры вытесненной задачи сохраняются на выходах S_preempted_valid, V_preempted_task_id, V_preempted_prior, V_preempted_wait, V_preempted_system, V_preempted_rem и V_preempted_started. Вытесненная задача возвращается в глобальную очередь с сохранением идентификатора, приоритета, накопленного времени ожидания, времени пребывания в системе, оставшегося времени обслуживания и признака ранее начатого выполнения. После этого более приоритетная задача передаётся на освободившийся процессор.

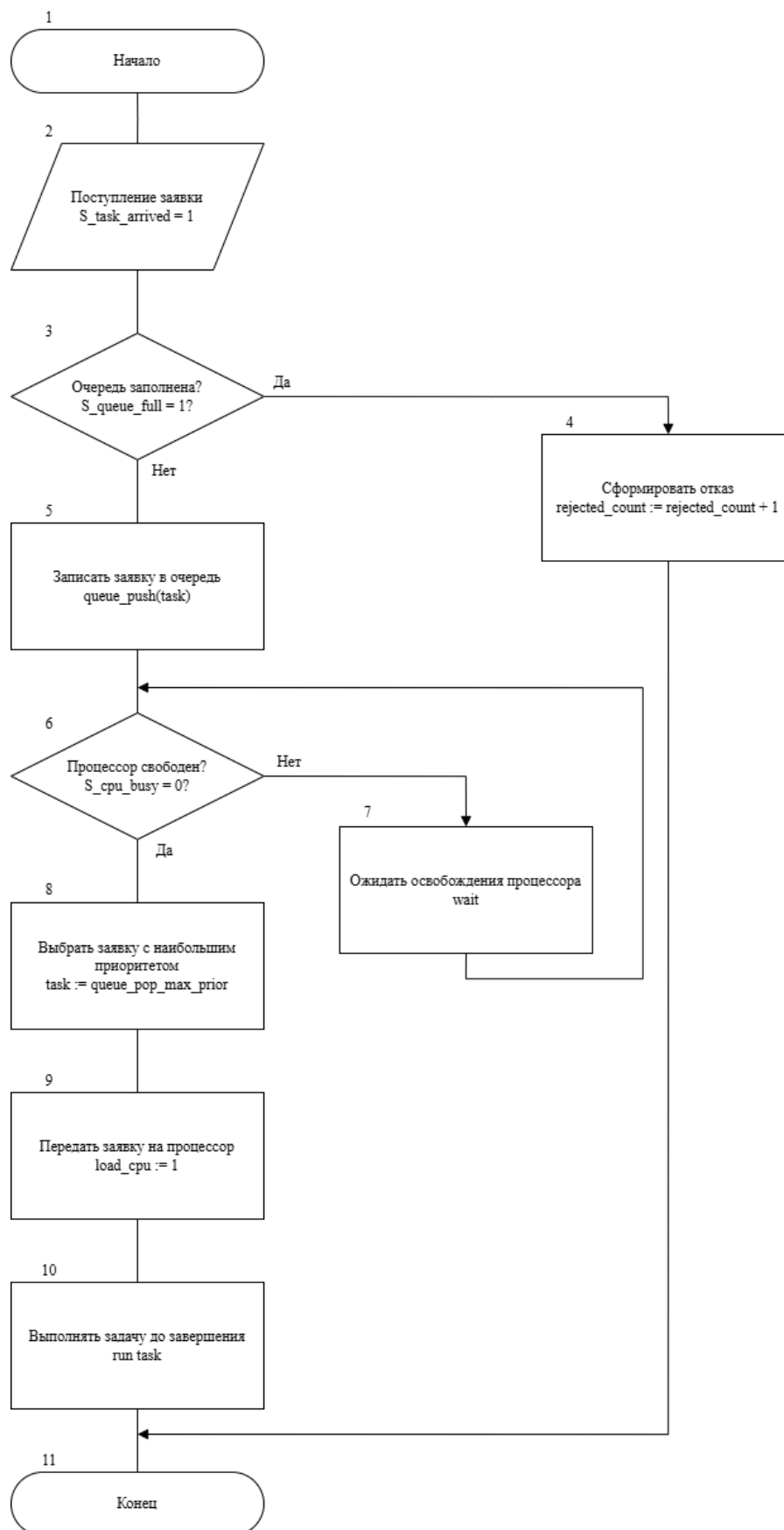


Рисунок 25 – Схема программы обслуживания задач в режиме относительных приоритетов

Схема программы обслуживания задач в режиме абсолютных приоритетов представлена на рисунке 26.

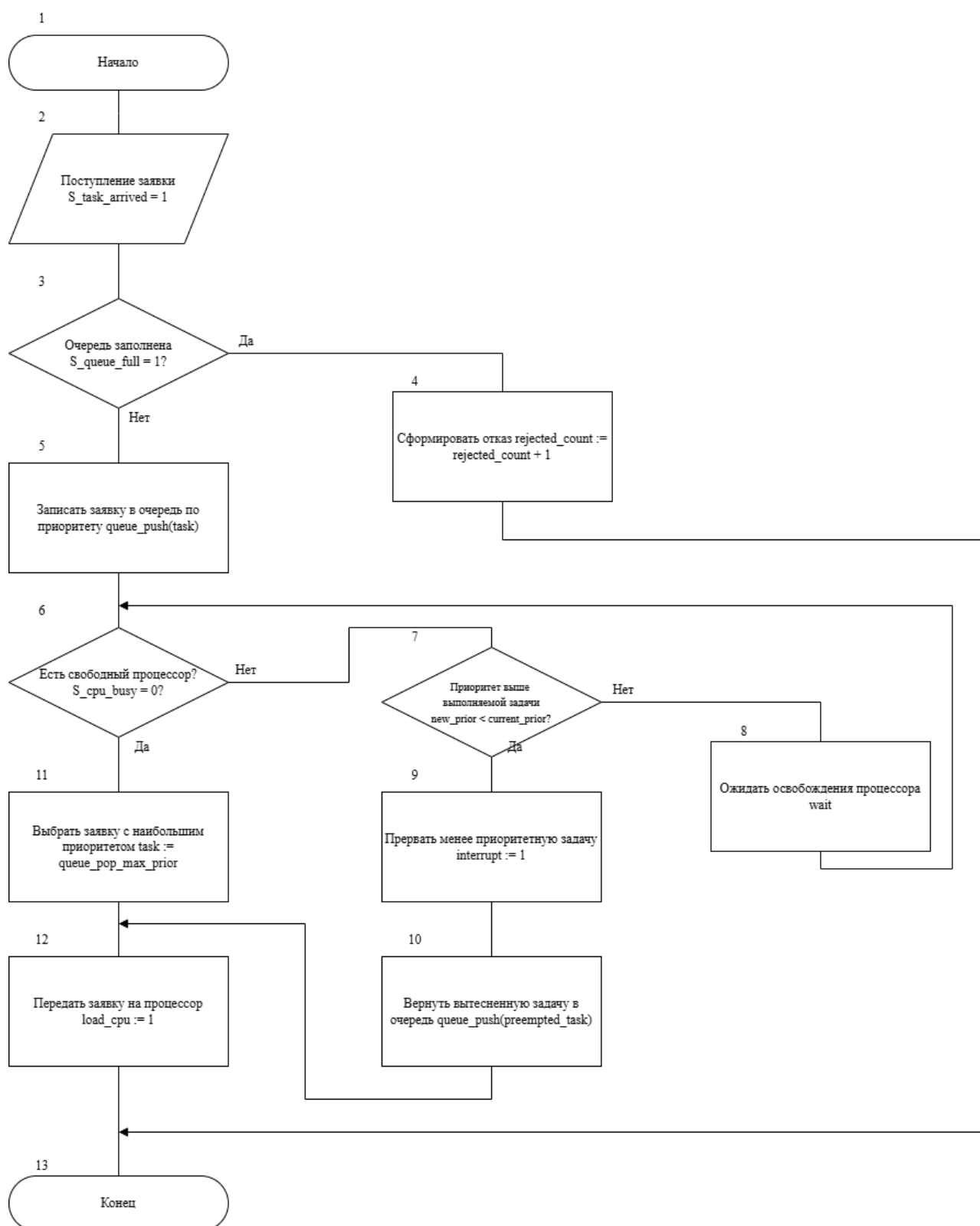


Рисунок 26 – Схема программы обслуживания задач в режиме абсолютных приоритетов

Блок сбора статистики предназначен для накопления результатов моделирования. Он получает сигналы создания задач, отказов, завершения обработки и текущей занятости процессоров. На основании этих данных подсчитываются общее число созданных задач, число отказов, количество завершённых задач, средние времена ожидания и пребывания в системе, а также показатели загрузки процессоров.

Структурная схема планировщика с глобальной очередью, включающая блок генерации задач, блок управления глобальной очередью, блок глобальной очереди, блок выбора процессора, блок синхронизации и назначения задач, процессорные блоки и блок сбора статистики, представлена на рисунке 27.

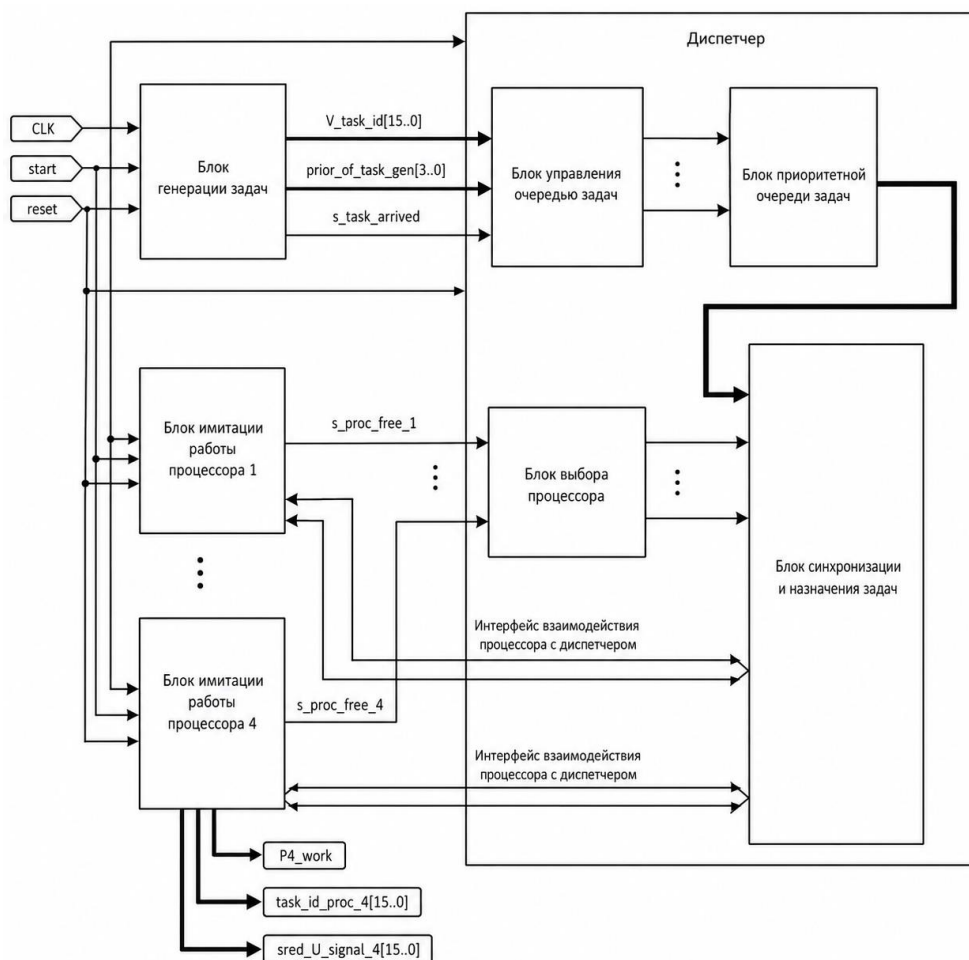


Рисунок 27 – Общий вид структурной схемы с глобальной приоритетной очередью

3.4 Моделирование аппаратного планировщика с локальными очередями

Модель аппаратного планировщика с локальными очередями была реализована на языке VHDL в виде набора программных модулей. Ввод описания, компиляция и имитационное моделирование выполнялись в среде Xilinx 9.1. Основу модели составляют блок генерации задач, блок выбора локальной очереди, блок диспетчеризации задач, четыре локальные очереди, четыре процессорных блока и блок сбора статистики.

Структурно модель планировщика с локальными очередями состоит из следующих основных блоков:

- блока генерации задач;
- блока выбора локальной очереди;
- блока диспетчеризации задач;
- локальных очередей;
- процессорных блоков;
- блока сбора статистики.

В отличие от модели с глобальной очередью, в данной структуре отсутствует единая очередь, общая для всех процессоров. Каждому процессору соответствует собственная локальная очередь. После поступления новая задача направляется в одну из локальных очередей, а дальнейший выбор задачи на обслуживание выполняется только в пределах очереди соответствующего процессора. Такой способ организации позволяет разделить входной поток между несколькими процессорными ветвями и уменьшить количество обращений к единому общему хранилищу задач.

Блок генерации задач формирует входной поток задач. Каждой новой задаче присваивается идентификатор и значение приоритета. Идентификатор передаётся по шине V_task_id, приоритет – по шине V_task_prior, а факт появления новой задачи фиксируется сигналом S_task_arrived. Интервалы между поступлениями задач формируются с использованием генератора случайных чисел и параметра

LAMBDA_C, задающего интенсивность входного потока. Приоритет задачи задаётся четырёхразрядным значением, что позволяет использовать шестнадцать уровней приоритета.

Блок выбора локальной очереди определяет, в какую из четырёх очередей должна быть помещена новая задача. Для этого анализируются текущие значения занятости локальных очередей V_use_1–V_use_4 и признаки их заполнения S_prior_full_1–S_prior_full_4. Выбор выполняется среди незаполненных очередей по минимальной текущей загрузке. Если несколько очередей имеют одинаковую минимальную загрузку, используется циклический принцип выбора, позволяющий распределять задачи между локальными очередями более равномерно.

Результат выбора фиксируется сигналами S_ocher_sel_1–S_ocher_sel_4 и кодом выбранной очереди V_selected_ocher. Если все локальные очереди заполнены, формируется признак S_all_queues_full. В этом случае задача не записывается в очередь и далее учитывается как отказ.

Схема программы выбора локальной очереди представлена на рисунке 28.

Блок диспетчеризации задач принимает сигнал появления новой задачи, её идентификатор и приоритет, а также результат выбора локальной очереди. Если выбранная очередь имеет свободное место, блок формирует один из сигналов записи S_wrreq_1–S_wrreq_4. Одновременно в выбранную очередь передаются параметры задачи: идентификатор, приоритет, начальное время ожидания, время пребывания в системе, оставшееся время обслуживания и признак начала выполнения. Если задача не может быть помещена ни в одну очередь, формируется сигнал отказа S_refuse или S_task_refused.

Каждая локальная очередь хранит задачи, ожидающие обслуживания соответствующим процессором. В составе записи очереди сохраняются идентификатор задачи, её приоритет, время ожидания, время пребывания в системе, оставшееся время обслуживания и признак того, что задача уже начинала выполняться.



Рисунок 28 – Схема программы выбора локальной очереди

Состояние локальной очереди определяется количеством находящихся в ней задач и признаком заполнения. Количество задач в очередях задаётся сигналами V_use_1–V_use_4, а признаки заполнения – сигналами S_prior_full_1–S_prior_full_4.

На выходе каждой локальной очереди формируются признак готовности задачи к передаче процессору и параметры этой задачи. При наличии задачи на выходе очереди формируется сигнал S_top_valid. Если соответствующий процессор свободен, задача передаётся на обслуживание. Удаление задачи из очереди выполняется сигналом S_por_top, а загрузка процессора – сигналом S_load_cpu.

Связь локальной очереди с процессорным блоком выполняется напрямую: каждая очередь обслуживает только свой процессор. На вход процессорного блока передаются идентификатор задачи, её приоритет, накопленное время ожидания, время пребывания в системе, оставшееся время обслуживания и признак повторного запуска задачи. Такой способ организации отличается от модели с глобальной очередью тем, что отсутствует отдельный этап выбора свободного процессора из общего набора процессоров.

Процессорный блок имитирует выполнение задачи. После получения сигнала загрузки процессор переходит из свободного состояния в состояние выполнения. Время обслуживания формируется случайным образом с использованием параметра MU_C. В процессе обслуживания уменьшается оставшееся время выполнения задачи и увеличивается время её пребывания в системе. После завершения обслуживания процессор формирует сигнал окончания задачи и снова переходит в свободное состояние. Состояние процессора характеризуется сигналами S_cpu_busy и S_cpu_running, а завершение обработки фиксируется сигналом S_task_done.

В рамках модели с локальными очередями реализованы три режима обслуживания задач: бесприоритетный режим, режим относительных приоритетов и режим абсолютных приоритетов.

В беспriorитетном режиме каждая локальная очередь работает по дисциплине FIFO. Новая задача записывается в конец выбранной очереди, а на обслуживание передаётся задача, находящаяся в начале этой очереди. Значение приоритета в данном режиме сохраняется вместе с задачей, но не влияет на порядок выбора. Такой режим используется как базовый вариант для сравнения с приоритетными дисциплинами обслуживания.

В режиме относительных приоритетов задачи внутри каждой локальной очереди упорядочиваются по значению приоритета. На обслуживание соответствующему процессору передаётся наиболее приоритетная задача из его локальной очереди. В VHDL-модели более высокий приоритет соответствует меньшему числовому значению поля `V_task_prior`. Если несколько задач имеют одинаковый приоритет, сохраняется порядок их поступления. Уже выполняемая задача в этом режиме не прерывается и занимает процессор до завершения обслуживания.

В режиме абсолютных приоритетов дополнительно реализован механизм вытеснения. Если в локальную очередь поступает задача с более высоким приоритетом, чем задача, выполняемая соответствующим процессором, формируется сигнал прерывания `S_interrupt`. При этом параметры вытесненной задачи сохраняются на выходах `S_preempted_valid`, `V_preempted_task_id`, `V_preempted_prior`, `V_preempted_wait`, `V_preempted_system`, `V_preempted_rem` и `V_preempted_started`. Затем вытесненная задача возвращается в свою локальную очередь с сохранением идентификатора, приоритета, накопленного времени ожидания, времени пребывания в системе, оставшегося времени обслуживания и признака ранее начатого выполнения. После этого более приоритетная задача передаётся на процессор.

Блок сбора статистики предназначен для накопления результатов моделирования. Он получает сигналы создания задач, отказов, завершения обработки и текущей занятости процессоров. На основании этих данных подсчитываются общее число созданных задач, число отказов, количество

завершённых задач, средние времена ожидания и пребывания в системе, а также показатели загрузки процессоров.

Структурная схема планировщика с локальными очередями, включающая блок генерации задач, блок выбора локальной очереди, блок диспетчеризации задач, локальные очереди, процессорные блоки и блок сбора статистики, представлена на рисунке 29.

Таким образом, VHDL-модель планировщика с локальными очередями реализует распределённый механизм хранения и обслуживания задач, при котором каждая процессорная ветвь имеет собственную очередь. Различие между режимами FIFO, относительных и абсолютных приоритетов заключается в правилах упорядочивания задач внутри локальных очередей и наличии механизма вытеснения в режиме абсолютных приоритетов. Полученная модель позволяет исследовать влияние локальной организации очередей на распределение нагрузки между процессорами, число отказов и временные характеристики обработки задач.

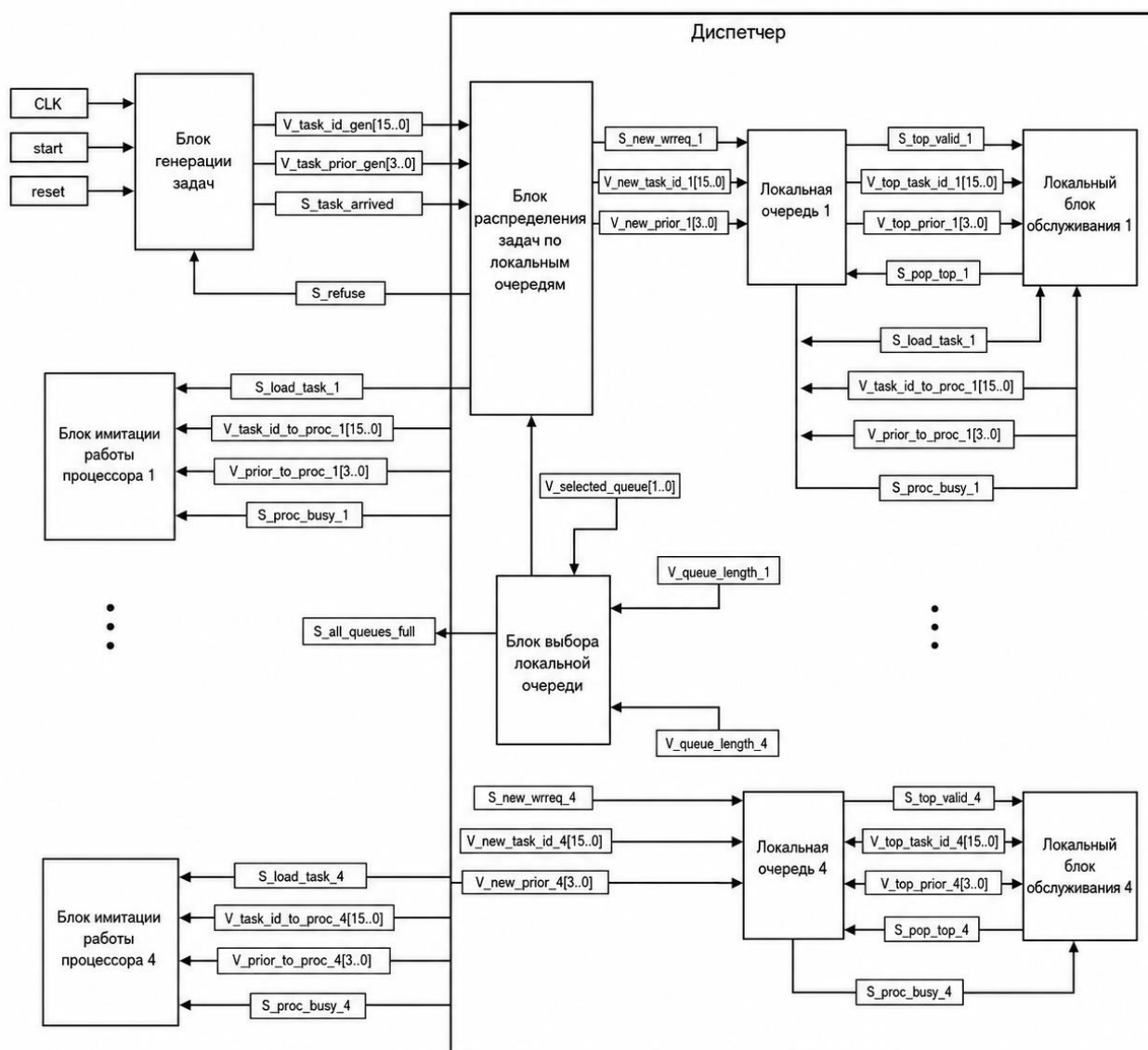


Рисунок 29 – Общий вид структурной схемы с локальными приоритетными очередями

3.5 Получение временных диаграмм и временных характеристик

Для разработанных моделей многопроцессорных планировщиков задач в среде Xilinx 9.1 были проведены имитационные эксперименты, в которых измерялись основные характеристики работы планировщиков: общее количество отказов, количество отказов по приоритетам, среднее время выполнения задачи в системе, среднее время выполнения задачи в системе по каждому приоритету,

среднее время отклика системы, а также среднее время отклика системы по каждому приоритету.

В работе рассматривались шесть вариантов планировщиков: планировщик с глобальной очередью без приоритетов, планировщик с глобальной очередью и относительными приоритетами, планировщик с глобальной очередью и абсолютными приоритетами, планировщик с локальными очередями без приоритетов, планировщик с локальными очередями и относительными приоритетами, а также планировщик с локальными очередями и абсолютными приоритетами. В моделях с глобальной очередью все поступающие задачи помещаются в единую очередь, из которой свободные процессоры выбирают задачи для выполнения. В моделях с локальными очередями каждая очередь связана с отдельным процессором, а поступающая задача распределяется между локальными очередями.

Для наглядного подтверждения имитационного характера разработанных моделей были получены временные диаграммы их работы. На рисунке 30 представлена временная диаграмма функционирования планировщика с глобальной очередью и абсолютными приоритетами. Диаграмма отражает поступление задач в систему, запись задач в глобальную очередь, выбор задачи из очереди, назначение задачи процессору, выполнение задачи и завершение обработки.

На временной диаграмме показано, что при поступлении задачи активируется сигнал `gen_task_arrived_int`, а её идентификатор и приоритет отображаются сигналами `gen_task_id` и `gen_task_prior`. Сигнал `new_wrrreq` показывает запись задачи в глобальную очередь, а `queue_length` отражает изменение количества задач в очереди. Сигналы `top_task_id_0`, `top_prior_0` и `pop_top_0` характеризуют выбор задачи из очереди и её снятие с верхней позиции. Далее задача назначается одному из процессоров, что отображается сигналом `assign_cpu_0` и импульсом `load_cpu_0_int`. Состояние процессора определяется сигналом `cpu_busy_0`, а текущая выполняемая задача и её приоритет отображаются

сигналами `current_task_id_0` и `current_prior_0`. Завершение обработки фиксируется сигналом `task_done_0`.

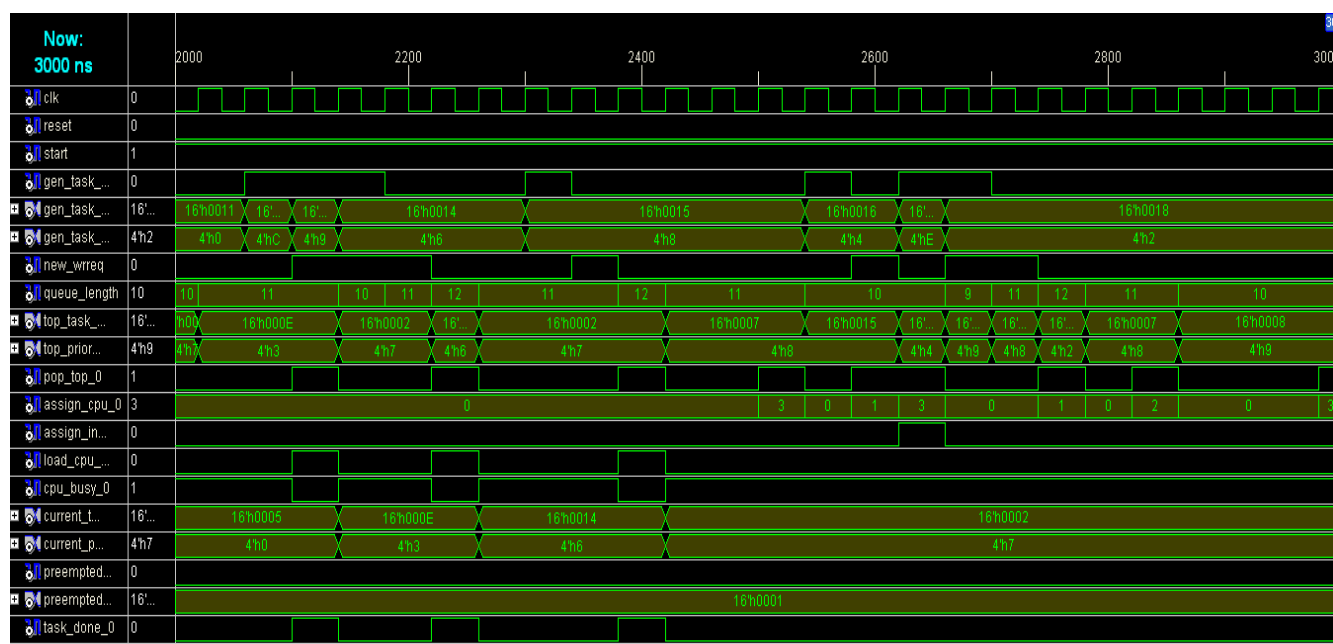


Рисунок 30 – Временная диаграмма работы модели планировщика с глобальной очередью и абсолютными приоритетами

На рисунке 31 представлена временная диаграмма функционирования планировщика с локальными очередями и абсолютными приоритетами. В отличие от модели с глобальной очередью, в данной модели поступающая задача сначала распределяется между локальными очередями, каждая из которых связана с отдельным процессором.

На временной диаграмме сигналы `gen_task_arrived_int`, `gen_task_id` и `gen_task_prior` отражают поступление задачи, её идентификатор и приоритет. Сигнал `selected_ocher` показывает номер локальной очереди, выбранной для размещения новой задачи. Сигнал `wreq_new[0:3]` отображает запись задач в локальные очереди, а `q_use[0:3]` показывает изменение их заполненности. Сигналы `q_top_task_id[0:3]` и `q_top_prior[0:3]` характеризуют задачи, находящиеся на выходах очередей, а `q_pop_top[0:3]` и `cpu_load[0:3]` показывают снятие задач из очередей и загрузку процессоров. Сигналы `cpu_busy[0:3]`, `cpu_current_id[0:3]` и

cpu_current_prior[0:3] отображают состояние процессоров и выполняемые ими задачи. Завершение обработки фиксируется сигналом task_done[0:3].

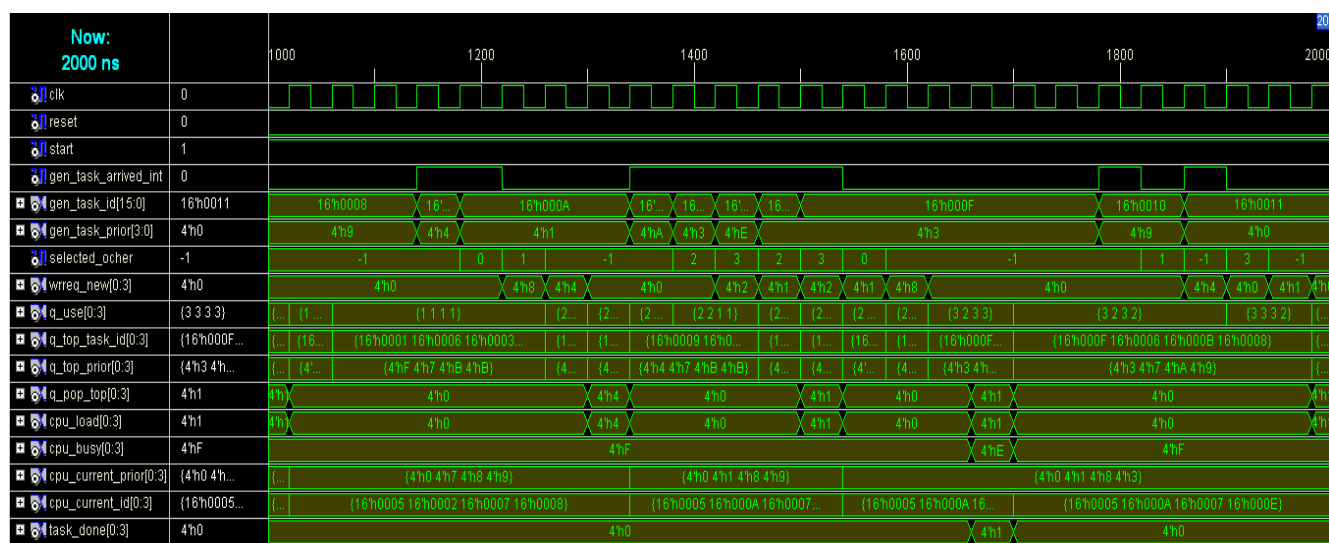


Рисунок 31 – Временная диаграмма работы модели планировщика с локальными очередями и абсолютными приоритетами

Для моделирования принято, что интенсивность входного потока задач изменяется при значениях $\lambda = 0,4; 0,3; 0,18; 0,125$ задач за один процессорный такт. Интервал между поступлениями задач в планировщик задавался экспоненциальным законом распределения. Значение интервала определялось по формуле (9):

$$x = -(1/\lambda) \cdot \ln(1 - s), \quad (9)$$

где s – значение равномерно распределённой в интервале (0; 1) случайной величины;

λ – интенсивность входного потока задач.

Среднее время обслуживания задачи в процессоре принято равным $v = 9,5$ такта. Число процессоров в многопроцессорной системе принято равным $n = 4$. Коэффициент загрузки системы определялся по формуле (10):

$$\rho = (\lambda \cdot v)/n, \quad (9)$$

где λ – интенсивность входного потока задач;

v – среднее время обслуживания задачи в процессоре;

n – число процессоров в системе.

Модельное время одного эксперимента составляло 3000000 тактов. При периоде синхросигнала 40 нс это позволило получить статистически устойчивые значения характеристик работы планировщиков при различных уровнях загрузки системы.

В ходе моделирования фиксировались среднее время выполнения задачи, среднее время отклика системы и количество отказов. Среднее время выполнения задачи определялось как полное время нахождения задачи в системе, то есть сумма времени ожидания в очереди и времени обработки в процессоре. Среднее время отклика системы определялось как время ожидания задачи в очереди до начала её обслуживания. Количество отказов определялось как число задач, которые не были приняты в очередь из-за отсутствия свободных мест. На рисунках 32–33 приведены результаты моделирования планировщика с глобальной очередью задач с абсолютным обслуживанием для $\rho = 0,95$ и $\rho = 0,71$.

 sr_vr_reb_sist	35.2998	 sr_vr_ojid_ocher	27.1065	 kol_sozd_zayavok	1269184
 sr_vr_reb_sist_0prior	9.05922	 sr_vr_ojid_ocher_0prior	1.00003	 kol_otkazov	95313
 sr_vr_reb_sist_1prior	9.11548	 sr_vr_ojid_ocher_1prior	1.003	 kol_otkazov_0prior	5931
 sr_vr_reb_sist_2prior	9.14483	 sr_vr_ojid_ocher_2prior	1.01481	 kol_otkazov_1prior	5906
 sr_vr_reb_sist_3prior	9.17013	 sr_vr_ojid_ocher_3prior	1.04524	 kol_otkazov_2prior	6038
 sr_vr_reb_sist_4prior	9.20587	 sr_vr_ojid_ocher_4prior	1.12981	 kol_otkazov_3prior	5973
 sr_vr_reb_sist_5prior	9.35166	 sr_vr_ojid_ocher_5prior	1.29511	 kol_otkazov_4prior	5995
 sr_vr_reb_sist_6prior	9.60903	 sr_vr_ojid_ocher_6prior	1.56679	 kol_otkazov_5prior	5969
 sr_vr_reb_sist_7prior	10.0136	 sr_vr_ojid_ocher_7prior	2.03695	 kol_otkazov_6prior	5990
 sr_vr_reb_sist_8prior	10.7899	 sr_vr_ojid_ocher_8prior	2.82881	 kol_otkazov_7prior	5949
 sr_vr_reb_sist_9prior	12.1721	 sr_vr_ojid_ocher_9prior	4.16446	 kol_otkazov_8prior	5870
 sr_vr_reb_sist_10prior	14.2594	 sr_vr_ojid_ocher_10prior	6.24208	 kol_otkazov_9prior	6003
 sr_vr_reb_sist_11prior	17.9006	 sr_vr_ojid_ocher_11prior	9.79413	 kol_otkazov_10prior	6040
 sr_vr_reb_sist_12prior	24.6234	 sr_vr_ojid_ocher_12prior	16.398	 kol_otkazov_11prior	5986
 sr_vr_reb_sist_13prior	38.3989	 sr_vr_ojid_ocher_13prior	29.9598	 kol_otkazov_12prior	5938
 sr_vr_reb_sist_14prior	73.4666	 sr_vr_ojid_ocher_14prior	64.7498	 kol_otkazov_13prior	5889
 sr_vr_reb_sist_15prior	294.223	 sr_vr_ojid_ocher_15prior	285.202	 kol_otkazov_14prior	5824
				 kol_otkazov_15prior	6012

Рисунок 32 – Временные характеристики планировщика с глобальной очередью задач с абсолютным обслуживанием в условиях $\rho = 0,95$

sr_vr_reb_sist	17.1828	sr_vr_ojid_ocher	8.92532	kol_sozd_zayavok	962019
sr_vr_reb_sist_0prior	9.10625	sr_vr_ojid_ocher_0prior	1.00003	kol_otkazov	4120
sr_vr_reb_sist_1prior	9.09739	sr_vr_ojid_ocher_1prior	1.00258	kol_otkazov_0prior	276
sr_vr_reb_sist_2prior	9.10311	sr_vr_ojid_ocher_2prior	1.01224	kol_otkazov_1prior	263
sr_vr_reb_sist_3prior	9.1562	sr_vr_ojid_ocher_3prior	1.04076	kol_otkazov_2prior	244
sr_vr_reb_sist_4prior	9.20993	sr_vr_ojid_ocher_4prior	1.11762	kol_otkazov_3prior	284
sr_vr_reb_sist_5prior	9.38068	sr_vr_ojid_ocher_5prior	1.2472	kol_otkazov_4prior	261
sr_vr_reb_sist_6prior	9.65352	sr_vr_ojid_ocher_6prior	1.52	kol_otkazov_5prior	224
sr_vr_reb_sist_7prior	10.0307	sr_vr_ojid_ocher_7prior	1.91654	kol_otkazov_6prior	244
sr_vr_reb_sist_8prior	10.719	sr_vr_ojid_ocher_8prior	2.57687	kol_otkazov_7prior	251
sr_vr_reb_sist_9prior	11.7656	sr_vr_ojid_ocher_9prior	3.55894	kol_otkazov_8prior	256
sr_vr_reb_sist_10prior	13.3021	sr_vr_ojid_ocher_10prior	5.11221	kol_otkazov_9prior	254
sr_vr_reb_sist_11prior	15.7923	sr_vr_ojid_ocher_11prior	7.46672	kol_otkazov_10prior	244
sr_vr_reb_sist_12prior	19.9582	sr_vr_ojid_ocher_12prior	11.5026	kol_otkazov_11prior	241
sr_vr_reb_sist_13prior	26.3759	sr_vr_ojid_ocher_13prior	17.8742	kol_otkazov_12prior	291
sr_vr_reb_sist_14prior	38.7119	sr_vr_ojid_ocher_14prior	30.015	kol_otkazov_13prior	255
sr_vr_reb_sist_15prior	63.2441	sr_vr_ojid_ocher_15prior	54.5263	kol_otkazov_14prior	264
				kol_otkazov_15prior	268

Рисунок 33 – Временные характеристики планировщика с глобальной очередью задач с абсолютным обслуживанием в условиях $\rho = 0,71$

На рисунках 34–35 приведены результаты моделирования планировщика с глобальной очередью задач с абсолютным обслуживанием для $\rho = 0,42$, $\rho = 0,71$.

sr_vr_reb_sist	9.76684	sr_vr_ojid_ocher	1.62641	kol_sozd_zayavok	572881
sr_vr_reb_sist_0prior	9.13857	sr_vr_ojid_ocher_0prior	1	kol_otkazov	0
sr_vr_reb_sist_1prior	9.04103	sr_vr_ojid_ocher_1prior	1.00006	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	9.11891	sr_vr_ojid_ocher_2prior	1.00179	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	9.13454	sr_vr_ojid_ocher_3prior	1.00762	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	9.11978	sr_vr_ojid_ocher_4prior	1.02024	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	9.11155	sr_vr_ojid_ocher_5prior	1.04961	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	9.17815	sr_vr_ojid_ocher_6prior	1.09424	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	9.25016	sr_vr_ojid_ocher_7prior	1.15521	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	9.34287	sr_vr_ojid_ocher_8prior	1.25054	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	9.49135	sr_vr_ojid_ocher_9prior	1.37276	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	9.65579	sr_vr_ojid_ocher_10prior	1.53182	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	10.0473	sr_vr_ojid_ocher_11prior	1.79756	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	10.2115	sr_vr_ojid_ocher_12prior	2.07117	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	10.7616	sr_vr_ojid_ocher_13prior	2.54551	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	11.3925	sr_vr_ojid_ocher_14prior	3.13831	kol_otkazov_13prior	0
sr_vr_reb_sist_15prior	12.2604	sr_vr_ojid_ocher_15prior	3.97381	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 34 – Временные характеристики планировщика с глобальной очередью задач с абсолютным обслуживанием в условиях $\rho = 0,42$

sr_vr_reb_sist	9.24695	sr_vr_ojid_ocher	1.1528	kol_sozd_zayavok	393500
sr_vr_reb_sist_0prior	9.12544	sr_vr_ojid_ocher_0prior	1	kol_otkazov	0
sr_vr_reb_sist_1prior	9.01491	sr_vr_ojid_ocher_1prior	1.00008	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	9.06604	sr_vr_ojid_ocher_2prior	1.00028	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	9.15094	sr_vr_ojid_ocher_3prior	1.00195	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	9.06891	sr_vr_ojid_ocher_4prior	1.00505	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	9.15884	sr_vr_ojid_ocher_5prior	1.01279	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	9.0528	sr_vr_ojid_ocher_6prior	1.0265	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	9.06217	sr_vr_ojid_ocher_7prior	1.03517	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	9.15809	sr_vr_ojid_ocher_8prior	1.06469	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	9.13964	sr_vr_ojid_ocher_9prior	1.08464	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	9.24555	sr_vr_ojid_ocher_10prior	1.13955	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	9.26111	sr_vr_ojid_ocher_11prior	1.19715	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	9.40439	sr_vr_ojid_ocher_12prior	1.27758	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	9.61674	sr_vr_ojid_ocher_13prior	1.37643	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	9.65626	sr_vr_ojid_ocher_14prior	1.52075	kol_otkazov_13prior	0
sr_vr_reb_sist_15prior	9.76039	sr_vr_ojid_ocher_15prior	1.69611	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 35 – Временные характеристики планировщика с глобальной очередью задач с абсолютным обслуживанием в условиях $\rho = 0,29$

На рисунках 36–37 приведены результаты моделирования планировщика с глобальной очередью задач с относительным обслуживанием для $\rho = 0,95$, $\rho = 0,71$.

sr_vr_reb_sist	33.737	sr_vr_ojid_ocher	23.853	kol_sozd_zayavok	1269185
sr_vr_reb_sist_0prior	11.9987	sr_vr_ojid_ocher_0prior	2.25374	kol_otkazov	108905
sr_vr_reb_sist_1prior	12.2951	sr_vr_ojid_ocher_1prior	2.49673	kol_otkazov_0prior	6752
sr_vr_reb_sist_2prior	12.5855	sr_vr_ojid_ocher_2prior	2.79166	kol_otkazov_1prior	6795
sr_vr_reb_sist_3prior	12.9181	sr_vr_ojid_ocher_3prior	3.16856	kol_otkazov_2prior	6959
sr_vr_reb_sist_4prior	13.3778	sr_vr_ojid_ocher_4prior	3.63839	kol_otkazov_3prior	6737
sr_vr_reb_sist_5prior	14.1075	sr_vr_ojid_ocher_5prior	4.25879	kol_otkazov_4prior	6800
sr_vr_reb_sist_6prior	14.7419	sr_vr_ojid_ocher_6prior	4.97243	kol_otkazov_5prior	6711
sr_vr_reb_sist_7prior	15.9554	sr_vr_ojid_ocher_7prior	6.02265	kol_otkazov_6prior	6844
sr_vr_reb_sist_8prior	17.3178	sr_vr_ojid_ocher_8prior	7.42058	kol_otkazov_7prior	6732
sr_vr_reb_sist_9prior	19.2	sr_vr_ojid_ocher_9prior	9.29792	kol_otkazov_8prior	6855
sr_vr_reb_sist_10prior	21.797	sr_vr_ojid_ocher_10prior	11.869	kol_otkazov_9prior	6736
sr_vr_reb_sist_11prior	26.2509	sr_vr_ojid_ocher_11prior	16.2963	kol_otkazov_10prior	6853
sr_vr_reb_sist_12prior	32.6727	sr_vr_ojid_ocher_12prior	22.6731	kol_otkazov_11prior	6796
sr_vr_reb_sist_13prior	45.1591	sr_vr_ojid_ocher_13prior	35.1389	kol_otkazov_12prior	6804
sr_vr_reb_sist_14prior	73.3485	sr_vr_ojid_ocher_14prior	63.339	kol_otkazov_13prior	6893
sr_vr_reb_sist_15prior	195.664	sr_vr_ojid_ocher_15prior	185.608	kol_otkazov_14prior	6774
				kol_otkazov_15prior	6864

Рисунок 36 – Временные характеристики в планировщике с глобальной очередью задач с относительным обслуживанием в условиях $\rho = 0,95$

 sr_vr_reb_sist	16.8241	 sr_vr_ojid_ocher	7.21521	 kol_sozd_zayavok	962019
 sr_vr_reb_sist_0prior	11.335	 sr_vr_ojid_ocher_0prior	1.82596	 kol_otkazov	3194
 sr_vr_reb_sist_1prior	11.5563	 sr_vr_ojid_ocher_1prior	1.96893	 kol_otkazov_0prior	213
 sr_vr_reb_sist_2prior	11.7472	 sr_vr_ojid_ocher_2prior	2.16025	 kol_otkazov_1prior	212
 sr_vr_reb_sist_3prior	11.9185	 sr_vr_ojid_ocher_3prior	2.36546	 kol_otkazov_2prior	212
 sr_vr_reb_sist_4prior	12.1252	 sr_vr_ojid_ocher_4prior	2.61983	 kol_otkazov_3prior	207
 sr_vr_reb_sist_5prior	12.5712	 sr_vr_ojid_ocher_5prior	2.98102	 kol_otkazov_4prior	184
 sr_vr_reb_sist_6prior	13.0032	 sr_vr_ojid_ocher_6prior	3.37103	 kol_otkazov_5prior	197
 sr_vr_reb_sist_7prior	13.4668	 sr_vr_ojid_ocher_7prior	3.87178	 kol_otkazov_6prior	196
 sr_vr_reb_sist_8prior	14.1099	 sr_vr_ojid_ocher_8prior	4.55062	 kol_otkazov_7prior	198
 sr_vr_reb_sist_9prior	14.9	 sr_vr_ojid_ocher_9prior	5.34956	 kol_otkazov_8prior	189
 sr_vr_reb_sist_10prior	16.0476	 sr_vr_ojid_ocher_10prior	6.4228	 kol_otkazov_9prior	203
 sr_vr_reb_sist_11prior	17.7036	 sr_vr_ojid_ocher_11prior	7.99699	 kol_otkazov_10prior	199
 sr_vr_reb_sist_12prior	19.7064	 sr_vr_ojid_ocher_12prior	10.0851	 kol_otkazov_11prior	180
 sr_vr_reb_sist_13prior	23.0141	 sr_vr_ojid_ocher_13prior	13.3313	 kol_otkazov_12prior	200
 sr_vr_reb_sist_14prior	28.2659	 sr_vr_ojid_ocher_14prior	18.5808	 kol_otkazov_13prior	197
 sr_vr_reb_sist_15prior	37.5903	 sr_vr_ojid_ocher_15prior	27.8381	 kol_otkazov_14prior	188
				 kol_otkazov_15prior	219

Рисунок 37 – Временные характеристики планировщика с глобальной очередью задач с относительным обслуживанием в условиях $\rho = 0,71$

На рисунках 38–39 приведены результаты моделирования планировщика с глобальной очередью задач с относительным обслуживанием для $\rho = 0,42$, $\rho = 0,29$.

 sr_vr_reb_sist	10.8638	 sr_vr_ojid_ocher	1.61577	 kol_sozd_zayavok	572881
 sr_vr_reb_sist_0prior	10.4738	 sr_vr_ojid_ocher_0prior	1.22656	 kol_otkazov	0
 sr_vr_reb_sist_1prior	10.3984	 sr_vr_ojid_ocher_1prior	1.25411	 kol_otkazov_0prior	0
 sr_vr_reb_sist_2prior	10.4478	 sr_vr_ojid_ocher_2prior	1.2801	 kol_otkazov_1prior	0
 sr_vr_reb_sist_3prior	10.5577	 sr_vr_ojid_ocher_3prior	1.30777	 kol_otkazov_2prior	0
 sr_vr_reb_sist_4prior	10.5506	 sr_vr_ojid_ocher_4prior	1.34431	 kol_otkazov_3prior	0
 sr_vr_reb_sist_5prior	10.6916	 sr_vr_ojid_ocher_5prior	1.38594	 kol_otkazov_4prior	0
 sr_vr_reb_sist_6prior	10.7826	 sr_vr_ojid_ocher_6prior	1.44771	 kol_otkazov_5prior	0
 sr_vr_reb_sist_7prior	10.7293	 sr_vr_ojid_ocher_7prior	1.49297	 kol_otkazov_6prior	0
 sr_vr_reb_sist_8prior	10.9041	 sr_vr_ojid_ocher_8prior	1.54089	 kol_otkazov_7prior	0
 sr_vr_reb_sist_9prior	10.9197	 sr_vr_ojid_ocher_9prior	1.64215	 kol_otkazov_8prior	0
 sr_vr_reb_sist_10prior	10.88	 sr_vr_ojid_ocher_10prior	1.70153	 kol_otkazov_9prior	0
 sr_vr_reb_sist_11prior	11.1275	 sr_vr_ojid_ocher_11prior	1.80002	 kol_otkazov_10prior	0
 sr_vr_reb_sist_12prior	11.1205	 sr_vr_ojid_ocher_12prior	1.89316	 kol_otkazov_11prior	0
 sr_vr_reb_sist_13prior	11.2144	 sr_vr_ojid_ocher_13prior	1.98262	 kol_otkazov_12prior	0
 sr_vr_reb_sist_14prior	11.424	 sr_vr_ojid_ocher_14prior	2.18999	 kol_otkazov_13prior	0
 sr_vr_reb_sist_15prior	11.595	 sr_vr_ojid_ocher_15prior	2.35724	 kol_otkazov_14prior	0
				 kol_otkazov_15prior	0

Рисунок 38 – Временные характеристики планировщика с глобальной очередью задач с относительным обслуживанием в условиях $\rho = 0,42$

sr_vr_reb_sist	10.311	sr_vr_ojid_ocher	1.15083	kol_sozd_zayavok	393500
sr_vr_reb_sist_0prior	10.3284	sr_vr_ojid_ocher_0prior	1.08264	kol_otkazov	0
sr_vr_reb_sist_1prior	10.3125	sr_vr_ojid_ocher_1prior	1.0848	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	10.3202	sr_vr_ojid_ocher_2prior	1.0854	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	10.2707	sr_vr_ojid_ocher_3prior	1.09796	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	10.2718	sr_vr_ojid_ocher_4prior	1.10053	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	10.2072	sr_vr_ojid_ocher_5prior	1.11782	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	10.2835	sr_vr_ojid_ocher_6prior	1.11779	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	10.323	sr_vr_ojid_ocher_7prior	1.13638	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	10.3065	sr_vr_ojid_ocher_8prior	1.15204	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	10.2606	sr_vr_ojid_ocher_9prior	1.16524	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	10.2964	sr_vr_ojid_ocher_10prior	1.18437	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	10.3391	sr_vr_ojid_ocher_11prior	1.19294	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	10.35	sr_vr_ojid_ocher_12prior	1.19718	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	10.384	sr_vr_ojid_ocher_13prior	1.21695	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	10.3546	sr_vr_ojid_ocher_14prior	1.23435	kol_otkazov_13prior	0
sr_vr_reb_sist_15prior	10.3676	sr_vr_ojid_ocher_15prior	1.24511	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 39 – Временные характеристики планировщика с глобальной очередью задач с относительным обслуживанием в условиях $\rho = 0,29$

На рисунках 40–41 приведены результаты моделирования планировщика с глобальной очередью задач с относительным обслуживанием для $\rho = 0,95$, $\rho = 0,71$.

sr_vr_reb_sist	32.5186	sr_vr_ojid_ocher	23.4961	kol_sozd_zayavok	1269185
sr_vr_reb_sist_0prior	32.3991	sr_vr_ojid_ocher_0prior	23.3942	kol_otkazov	110185
sr_vr_reb_sist_1prior	32.5342	sr_vr_ojid_ocher_1prior	23.5346	kol_otkazov_0prior	6925
sr_vr_reb_sist_2prior	32.5272	sr_vr_ojid_ocher_2prior	23.4955	kol_otkazov_1prior	6864
sr_vr_reb_sist_3prior	32.5498	sr_vr_ojid_ocher_3prior	23.4861	kol_otkazov_2prior	7015
sr_vr_reb_sist_4prior	32.4779	sr_vr_ojid_ocher_4prior	23.4637	kol_otkazov_3prior	6887
sr_vr_reb_sist_5prior	32.493	sr_vr_ojid_ocher_5prior	23.5282	kol_otkazov_4prior	6990
sr_vr_reb_sist_6prior	32.5249	sr_vr_ojid_ocher_6prior	23.4858	kol_otkazov_5prior	7020
sr_vr_reb_sist_7prior	32.4915	sr_vr_ojid_ocher_7prior	23.489	kol_otkazov_6prior	6913
sr_vr_reb_sist_8prior	32.4592	sr_vr_ojid_ocher_8prior	23.4332	kol_otkazov_7prior	6775
sr_vr_reb_sist_9prior	32.6311	sr_vr_ojid_ocher_9prior	23.55	kol_otkazov_8prior	6941
sr_vr_reb_sist_10prior	32.6364	sr_vr_ojid_ocher_10prior	23.5453	kol_otkazov_9prior	6784
sr_vr_reb_sist_11prior	32.5966	sr_vr_ojid_ocher_11prior	23.5935	kol_otkazov_10prior	6899
sr_vr_reb_sist_12prior	32.4313	sr_vr_ojid_ocher_12prior	23.4386	kol_otkazov_11prior	6686
sr_vr_reb_sist_13prior	32.5728	sr_vr_ojid_ocher_13prior	23.5296	kol_otkazov_12prior	6786
sr_vr_reb_sist_14prior	32.4325	sr_vr_ojid_ocher_14prior	23.4277	kol_otkazov_13prior	6878
sr_vr_reb_sist_15prior	32.541	sr_vr_ojid_ocher_15prior	23.5435	kol_otkazov_14prior	6874
				kol_otkazov_15prior	6948

Рисунок 40 – Временные характеристики планировщика с глобальной очередью задач с беспriorитетным обслуживанием в условиях $\rho = 0,95$

sr_vr_reb_sist	15.944	sr_vr_ojid_ocher	7.27254	kol_sozd_zayavok	962019
sr_vr_reb_sist_0prior	16.0266	sr_vr_ojid_ocher_0prior	7.29593	kol_otkazov	3590
sr_vr_reb_sist_1prior	15.9987	sr_vr_ojid_ocher_1prior	7.27456	kol_otkazov_0prior	229
sr_vr_reb_sist_2prior	15.9609	sr_vr_ojid_ocher_2prior	7.26243	kol_otkazov_1prior	225
sr_vr_reb_sist_3prior	15.95	sr_vr_ojid_ocher_3prior	7.28729	kol_otkazov_2prior	226
sr_vr_reb_sist_4prior	15.838	sr_vr_ojid_ocher_4prior	7.23762	kol_otkazov_3prior	234
sr_vr_reb_sist_5prior	15.9821	sr_vr_ojid_ocher_5prior	7.30282	kol_otkazov_4prior	200
sr_vr_reb_sist_6prior	15.9247	sr_vr_ojid_ocher_6prior	7.2589	kol_otkazov_5prior	214
sr_vr_reb_sist_7prior	15.8676	sr_vr_ojid_ocher_7prior	7.18122	kol_otkazov_6prior	221
sr_vr_reb_sist_8prior	15.9098	sr_vr_ojid_ocher_8prior	7.21701	kol_otkazov_7prior	220
sr_vr_reb_sist_9prior	16.0325	sr_vr_ojid_ocher_9prior	7.33112	kol_otkazov_8prior	223
sr_vr_reb_sist_10prior	15.9782	sr_vr_ojid_ocher_10prior	7.29061	kol_otkazov_9prior	261
sr_vr_reb_sist_11prior	15.9643	sr_vr_ojid_ocher_11prior	7.31422	kol_otkazov_10prior	201
sr_vr_reb_sist_12prior	15.9647	sr_vr_ojid_ocher_12prior	7.30876	kol_otkazov_11prior	238
sr_vr_reb_sist_13prior	15.8655	sr_vr_ojid_ocher_13prior	7.25457	kol_otkazov_12prior	217
sr_vr_reb_sist_14prior	15.8687	sr_vr_ojid_ocher_14prior	7.22118	kol_otkazov_13prior	232
sr_vr_reb_sist_15prior	15.9726	sr_vr_ojid_ocher_15prior	7.32252	kol_otkazov_14prior	215
				kol_otkazov_15prior	234

Рисунок 41 – Временные характеристики планировщика с глобальной очередью задач с бесприоритетным обслуживанием в условиях $p = 0,71$

На рисунках 42–43 приведены результаты моделирования планировщика с глобальной очередью задач с относительным обслуживанием для $p = 0,42$, $p = 0,29$.

sr_vr_reb_sist	9.87454	sr_vr_ojid_ocher	1.62671	kol_sozd_zayavok	57288
sr_vr_reb_sist_0prior	9.84602	sr_vr_ojid_ocher_0prior	1.62693	kol_otkazov	0
sr_vr_reb_sist_1prior	9.92155	sr_vr_ojid_ocher_1prior	1.63785	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	9.83823	sr_vr_ojid_ocher_2prior	1.63471	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	9.90854	sr_vr_ojid_ocher_3prior	1.61565	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	9.89321	sr_vr_ojid_ocher_4prior	1.66219	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	9.83611	sr_vr_ojid_ocher_5prior	1.61043	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	9.8907	sr_vr_ojid_ocher_6prior	1.60747	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	9.85704	sr_vr_ojid_ocher_7prior	1.6453	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	9.828	sr_vr_ojid_ocher_8prior	1.63531	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	9.93075	sr_vr_ojid_ocher_9prior	1.62497	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	9.86748	sr_vr_ojid_ocher_10prior	1.61952	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	9.8913	sr_vr_ojid_ocher_11prior	1.63875	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	9.85217	sr_vr_ojid_ocher_12prior	1.62513	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	9.84222	sr_vr_ojid_ocher_13prior	1.60862	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	9.95665	sr_vr_ojid_ocher_14prior	1.631	kol_otkazov_13prior	0
sr_vr_reb_sist_15prior	9.83359	sr_vr_ojid_ocher_15prior	1.60309	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 42 – Временные характеристики планировщика с глобальной очередью задач с бесприоритетным обслуживанием в условиях $p = 0,42$

 sr_vr_reb_sist	9.29479	 sr_vr_ojid_ocher	1.16387	 kol_sozd_zayavok	393500
 sr_vr_reb_sist_0prior	9.3261	 sr_vr_ojid_ocher_0prior	1.16641	 kol_otkazov	0
 sr_vr_reb_sist_1prior	9.27065	 sr_vr_ojid_ocher_1prior	1.16877	 kol_otkazov_0prior	0
 sr_vr_reb_sist_2prior	9.30049	 sr_vr_ojid_ocher_2prior	1.15779	 kol_otkazov_1prior	0
 sr_vr_reb_sist_3prior	9.36991	 sr_vr_ojid_ocher_3prior	1.17162	 kol_otkazov_2prior	0
 sr_vr_reb_sist_4prior	9.20224	 sr_vr_ojid_ocher_4prior	1.16976	 kol_otkazov_3prior	0
 sr_vr_reb_sist_5prior	9.26804	 sr_vr_ojid_ocher_5prior	1.16228	 kol_otkazov_4prior	0
 sr_vr_reb_sist_6prior	9.29838	 sr_vr_ojid_ocher_6prior	1.16133	 kol_otkazov_5prior	0
 sr_vr_reb_sist_7prior	9.32721	 sr_vr_ojid_ocher_7prior	1.16846	 kol_otkazov_6prior	0
 sr_vr_reb_sist_8prior	9.31635	 sr_vr_ojid_ocher_8prior	1.17114	 kol_otkazov_7prior	0
 sr_vr_reb_sist_9prior	9.19586	 sr_vr_ojid_ocher_9prior	1.16222	 kol_otkazov_8prior	0
 sr_vr_reb_sist_10prior	9.30791	 sr_vr_ojid_ocher_10prior	1.16305	 kol_otkazov_9prior	0
 sr_vr_reb_sist_11prior	9.31568	 sr_vr_ojid_ocher_11prior	1.16558	 kol_otkazov_10prior	0
 sr_vr_reb_sist_12prior	9.2697	 sr_vr_ojid_ocher_12prior	1.15637	 kol_otkazov_11prior	0
 sr_vr_reb_sist_13prior	9.35518	 sr_vr_ojid_ocher_13prior	1.14337	 kol_otkazov_12prior	0
 sr_vr_reb_sist_14prior	9.25796	 sr_vr_ojid_ocher_14prior	1.16583	 kol_otkazov_13prior	0
 sr_vr_reb_sist_15prior	9.33461	 sr_vr_ojid_ocher_15prior	1.1681	 kol_otkazov_14prior	0
				 kol_otkazov_15prior	0

Рисунок 43 – Временные характеристики планировщика с глобальной очередью задач с беспriorитетным обслуживанием в условиях $\rho = 0,29$

На рисунках 44–45 приведены результаты моделирования планировщика с локальными очередями задач с абсолютным обслуживанием для $\rho = 0,95$, $\rho = 0,71$.

 sr_vr_reb_sist	35.3498	 sr_vr_ojid_ocher	26.3451	 kol_sozd_zayavok	1269183
 sr_vr_reb_sist_0prior	10.4998	 sr_vr_ojid_ocher_0prior	1.36754	 kol_otkazov	114049
 sr_vr_reb_sist_1prior	11.142	 sr_vr_ojid_ocher_1prior	2.05376	 kol_otkazov_0prior	7101
 sr_vr_reb_sist_2prior	11.8972	 sr_vr_ojid_ocher_2prior	2.8422	 kol_otkazov_1prior	7190
 sr_vr_reb_sist_3prior	12.7751	 sr_vr_ojid_ocher_3prior	3.71993	 kol_otkazov_2prior	7121
 sr_vr_reb_sist_4prior	13.7491	 sr_vr_ojid_ocher_4prior	4.67778	 kol_otkazov_3prior	7149
 sr_vr_reb_sist_5prior	14.9338	 sr_vr_ojid_ocher_5prior	5.9567	 kol_otkazov_4prior	7093
 sr_vr_reb_sist_6prior	16.3024	 sr_vr_ojid_ocher_6prior	7.36146	 kol_otkazov_5prior	7154
 sr_vr_reb_sist_7prior	17.7952	 sr_vr_ojid_ocher_7prior	8.82284	 kol_otkazov_6prior	6985
 sr_vr_reb_sist_8prior	19.9993	 sr_vr_ojid_ocher_8prior	10.953	 kol_otkazov_7prior	7159
 sr_vr_reb_sist_9prior	22.6669	 sr_vr_ojid_ocher_9prior	13.6603	 kol_otkazov_8prior	7087
 sr_vr_reb_sist_10prior	26.1243	 sr_vr_ojid_ocher_10prior	17.1612	 kol_otkazov_9prior	7166
 sr_vr_reb_sist_11prior	30.892	 sr_vr_ojid_ocher_11prior	21.9541	 kol_otkazov_10prior	7301
 sr_vr_reb_sist_12prior	38.2163	 sr_vr_ojid_ocher_12prior	29.338	 kol_otkazov_11prior	7106
 sr_vr_reb_sist_13prior	51.017	 sr_vr_ojid_ocher_13prior	42.0951	 kol_otkazov_12prior	7193
 sr_vr_reb_sist_14prior	77.3403	 sr_vr_ojid_ocher_14prior	68.4225	 kol_otkazov_13prior	6999
 sr_vr_reb_sist_15prior	187.898	 sr_vr_ojid_ocher_15prior	178.796	 kol_otkazov_14prior	7130
				 kol_otkazov_15prior	7115

Рисунок 44 – Временные характеристики планировщика с локальными очередями задач с абсолютным обслуживанием в условиях $\rho = 0,95$

sr_vr_reb_sist	21.4014	sr_vr_ojid_ocher	12.3211	kol_sozd_zayavok	962019
sr_vr_reb_sist_0prior	10.4147	sr_vr_ojid_ocher_0prior	1.29316	kol_otkazov	6722
sr_vr_reb_sist_1prior	10.8715	sr_vr_ojid_ocher_1prior	1.85718	kol_otkazov_0prior	388
sr_vr_reb_sist_2prior	11.5872	sr_vr_ojid_ocher_2prior	2.48076	kol_otkazov_1prior	416
sr_vr_reb_sist_3prior	12.3226	sr_vr_ojid_ocher_3prior	3.2418	kol_otkazov_2prior	436
sr_vr_reb_sist_4prior	13.0971	sr_vr_ojid_ocher_4prior	3.96982	kol_otkazov_3prior	414
sr_vr_reb_sist_5prior	13.8835	sr_vr_ojid_ocher_5prior	4.8623	kol_otkazov_4prior	453
sr_vr_reb_sist_6prior	15.0049	sr_vr_ojid_ocher_6prior	5.93782	kol_otkazov_5prior	429
sr_vr_reb_sist_7prior	16.1753	sr_vr_ojid_ocher_7prior	7.09315	kol_otkazov_6prior	383
sr_vr_reb_sist_8prior	17.7028	sr_vr_ojid_ocher_8prior	8.59302	kol_otkazov_7prior	418
sr_vr_reb_sist_9prior	19.3921	sr_vr_ojid_ocher_9prior	10.2705	kol_otkazov_8prior	403
sr_vr_reb_sist_10prior	21.4543	sr_vr_ojid_ocher_10prior	12.4072	kol_otkazov_9prior	433
sr_vr_reb_sist_11prior	24.1614	sr_vr_ojid_ocher_11prior	15.0411	kol_otkazov_10prior	382
sr_vr_reb_sist_12prior	27.6308	sr_vr_ojid_ocher_12prior	18.556	kol_otkazov_11prior	398
sr_vr_reb_sist_13prior	32.6319	sr_vr_ojid_ocher_13prior	23.5979	kol_otkazov_12prior	463
sr_vr_reb_sist_14prior	40.4858	sr_vr_ojid_ocher_14prior	31.4422	kol_otkazov_13prior	414
sr_vr_reb_sist_15prior	55.3714	sr_vr_ojid_ocher_15prior	46.2606	kol_otkazov_14prior	433
				kol_otkazov_15prior	459

Рисунок 45 – Временные характеристики планировщика с локальными очередями задач с абсолютным обслуживанием в условиях $\rho = 0,71$

На рисунках 46–47 приведены результаты моделирования планировщика с локальными очередями задач с абсолютным обслуживанием для $\rho = 0,42$, $\rho = 0,29$.

sr_vr_reb_sist	12.6527	sr_vr_ojid_ocher	3.5537	kol_otkazov	3
sr_vr_reb_sist_0prior	10.1669	sr_vr_ojid_ocher_0prior	1.09607	kol_otkazov_0prior	0
sr_vr_reb_sist_1prior	10.3761	sr_vr_ojid_ocher_1prior	1.31785	kol_otkazov_1prior	0
sr_vr_reb_sist_2prior	10.5855	sr_vr_ojid_ocher_2prior	1.56263	kol_otkazov_2prior	0
sr_vr_reb_sist_3prior	10.9176	sr_vr_ojid_ocher_3prior	1.85689	kol_otkazov_3prior	0
sr_vr_reb_sist_4prior	11.2304	sr_vr_ojid_ocher_4prior	2.14603	kol_otkazov_4prior	2
sr_vr_reb_sist_5prior	11.511	sr_vr_ojid_ocher_5prior	2.39819	kol_otkazov_5prior	0
sr_vr_reb_sist_6prior	11.7241	sr_vr_ojid_ocher_6prior	2.67344	kol_otkazov_6prior	0
sr_vr_reb_sist_7prior	12.0728	sr_vr_ojid_ocher_7prior	2.99821	kol_otkazov_7prior	0
sr_vr_reb_sist_8prior	12.5406	sr_vr_ojid_ocher_8prior	3.40553	kol_otkazov_8prior	0
sr_vr_reb_sist_9prior	13.0373	sr_vr_ojid_ocher_9prior	3.85912	kol_otkazov_9prior	0
sr_vr_reb_sist_10prior	13.3476	sr_vr_ojid_ocher_10prior	4.25471	kol_otkazov_10prior	0
sr_vr_reb_sist_11prior	13.8115	sr_vr_ojid_ocher_11prior	4.73268	kol_otkazov_11prior	0
sr_vr_reb_sist_12prior	14.2936	sr_vr_ojid_ocher_12prior	5.15674	kol_otkazov_12prior	0
sr_vr_reb_sist_13prior	14.8743	sr_vr_ojid_ocher_13prior	5.80871	kol_otkazov_13prior	1
sr_vr_reb_sist_14prior	15.6443	sr_vr_ojid_ocher_14prior	6.46294	kol_otkazov_14prior	0
sr_vr_reb_sist_15prior	16.2766	sr_vr_ojid_ocher_15prior	7.09726	kol_otkazov_15prior	0

Рисунок 46 – Временные характеристики планировщика с локальными очередями с абсолютным обслуживанием в условиях $\rho = 0,42$

 sr_vr_reb_sist	11.2141	 sr_vr_ojid_ocher	2.13289	 kol_sozd_zayavok	393500
 sr_vr_reb_sist_0prior	10.0642	 sr_vr_ojid_ocher_0prior	1.05779	 kol_otkazov	0
 sr_vr_reb_sist_1prior	10.2798	 sr_vr_ojid_ocher_1prior	1.18512	 kol_otkazov_0prior	0
 sr_vr_reb_sist_2prior	10.3595	 sr_vr_ojid_ocher_2prior	1.27105	 kol_otkazov_1prior	0
 sr_vr_reb_sist_3prior	10.4875	 sr_vr_ojid_ocher_3prior	1.41649	 kol_otkazov_2prior	0
 sr_vr_reb_sist_4prior	10.7574	 sr_vr_ojid_ocher_4prior	1.56896	 kol_otkazov_3prior	0
 sr_vr_reb_sist_5prior	10.8786	 sr_vr_ojid_ocher_5prior	1.72346	 kol_otkazov_4prior	0
 sr_vr_reb_sist_6prior	10.8722	 sr_vr_ojid_ocher_6prior	1.87729	 kol_otkazov_5prior	0
 sr_vr_reb_sist_7prior	11.0779	 sr_vr_ojid_ocher_7prior	1.98678	 kol_otkazov_6prior	0
 sr_vr_reb_sist_8prior	11.1827	 sr_vr_ojid_ocher_8prior	2.1268	 kol_otkazov_7prior	0
 sr_vr_reb_sist_9prior	11.3328	 sr_vr_ojid_ocher_9prior	2.30319	 kol_otkazov_8prior	0
 sr_vr_reb_sist_10prior	11.7009	 sr_vr_ojid_ocher_10prior	2.48701	 kol_otkazov_9prior	0
 sr_vr_reb_sist_11prior	11.6972	 sr_vr_ojid_ocher_11prior	2.63346	 kol_otkazov_10prior	0
 sr_vr_reb_sist_12prior	11.9267	 sr_vr_ojid_ocher_12prior	2.86783	 kol_otkazov_11prior	0
 sr_vr_reb_sist_13prior	12.0422	 sr_vr_ojid_ocher_13prior	2.97058	 kol_otkazov_12prior	0
 sr_vr_reb_sist_14prior	12.1616	 sr_vr_ojid_ocher_14prior	3.157	 kol_otkazov_13prior	0
 sr_vr_reb_sist_15prior	12.5789	 sr_vr_ojid_ocher_15prior	3.47122	 kol_otkazov_14prior	0
				 kol_otkazov_15prior	0

Рисунок 47 – Временные характеристики планировщика с локальными очередями задач с абсолютным обслуживанием в условиях $\rho = 0,29$

На рисунках 48–49 приведены результаты моделирования планировщика с локальными очередями задач с относительным обслуживанием для $\rho = 0,95$, $\rho = 0,71$.

 sr_vr_reb_sist	33.197	 sr_vr_ojid_ocher	24.102	 kol_sozd_zayavok	1269183
 sr_vr_reb_sist_0prior	18.5066	 sr_vr_ojid_ocher_0prior	9.45025	 kol_otkazov	82731
 sr_vr_reb_sist_1prior	19.1149	 sr_vr_ojid_ocher_1prior	10.022	 kol_otkazov_0prior	5184
 sr_vr_reb_sist_2prior	19.8585	 sr_vr_ojid_ocher_2prior	10.7298	 kol_otkazov_1prior	5233
 sr_vr_reb_sist_3prior	20.5667	 sr_vr_ojid_ocher_3prior	11.5241	 kol_otkazov_2prior	5116
 sr_vr_reb_sist_4prior	21.4796	 sr_vr_ojid_ocher_4prior	12.3801	 kol_otkazov_3prior	5245
 sr_vr_reb_sist_5prior	22.5265	 sr_vr_ojid_ocher_5prior	13.4386	 kol_otkazov_4prior	5231
 sr_vr_reb_sist_6prior	23.6336	 sr_vr_ojid_ocher_6prior	14.5536	 kol_otkazov_5prior	5105
 sr_vr_reb_sist_7prior	24.8854	 sr_vr_ojid_ocher_7prior	15.7843	 kol_otkazov_6prior	5060
 sr_vr_reb_sist_8prior	26.7435	 sr_vr_ojid_ocher_8prior	17.6335	 kol_otkazov_7prior	5219
 sr_vr_reb_sist_9prior	28.6938	 sr_vr_ojid_ocher_9prior	19.6095	 kol_otkazov_8prior	5267
 sr_vr_reb_sist_10prior	31.1507	 sr_vr_ojid_ocher_10prior	22.1079	 kol_otkazov_9prior	5114
 sr_vr_reb_sist_11prior	34.4572	 sr_vr_ojid_ocher_11prior	25.3614	 kol_otkazov_10prior	5057
 sr_vr_reb_sist_12prior	39.2228	 sr_vr_ojid_ocher_12prior	30.1859	 kol_otkazov_11prior	5085
 sr_vr_reb_sist_13prior	46.7008	 sr_vr_ojid_ocher_13prior	37.5998	 kol_otkazov_12prior	5214
 sr_vr_reb_sist_14prior	60.0124	 sr_vr_ojid_ocher_14prior	50.8485	 kol_otkazov_13prior	5261
 sr_vr_reb_sist_15prior	93.4215	 sr_vr_ojid_ocher_15prior	84.2264	 kol_otkazov_14prior	5086
				 kol_otkazov_15prior	5254

Рисунок 48 – Временные характеристики планировщика с локальными очередями задач с относительным обслуживанием в условиях $\rho = 0,95$

sr_vr_reb_sist	20.1504	sr_vr_ojid_ocher	11.0616	kol_sozd_zayavok	962019
sr_vr_reb_sist_0prior	16.0722	sr_vr_ojid_ocher_0prior	6.96204	kol_otkazov	3785
sr_vr_reb_sist_1prior	16.3956	sr_vr_ojid_ocher_1prior	7.28372	kol_otkazov_0prior	237
sr_vr_reb_sist_2prior	16.6179	sr_vr_ojid_ocher_2prior	7.50339	kol_otkazov_1prior	216
sr_vr_reb_sist_3prior	16.9463	sr_vr_ojid_ocher_3prior	7.88239	kol_otkazov_2prior	252
sr_vr_reb_sist_4prior	17.321	sr_vr_ojid_ocher_4prior	8.27173	kol_otkazov_3prior	245
sr_vr_reb_sist_5prior	17.8623	sr_vr_ojid_ocher_5prior	8.76636	kol_otkazov_4prior	257
sr_vr_reb_sist_6prior	18.2253	sr_vr_ojid_ocher_6prior	9.12076	kol_otkazov_5prior	237
sr_vr_reb_sist_7prior	18.6796	sr_vr_ojid_ocher_7prior	9.6153	kol_otkazov_6prior	210
sr_vr_reb_sist_8prior	19.2555	sr_vr_ojid_ocher_8prior	10.1719	kol_otkazov_7prior	241
sr_vr_reb_sist_9prior	19.9815	sr_vr_ojid_ocher_9prior	10.9453	kol_otkazov_8prior	226
sr_vr_reb_sist_10prior	20.8525	sr_vr_ojid_ocher_10prior	11.7281	kol_otkazov_9prior	233
sr_vr_reb_sist_11prior	21.6702	sr_vr_ojid_ocher_11prior	12.575	kol_otkazov_10prior	263
sr_vr_reb_sist_12prior	22.6194	sr_vr_ojid_ocher_12prior	13.5535	kol_otkazov_11prior	236
sr_vr_reb_sist_13prior	24.3994	sr_vr_ojid_ocher_13prior	15.2977	kol_otkazov_12prior	224
sr_vr_reb_sist_14prior	26.1164	sr_vr_ojid_ocher_14prior	17.0703	kol_otkazov_13prior	229
sr_vr_reb_sist_15prior	29.3248	sr_vr_ojid_ocher_15prior	20.1728	kol_otkazov_14prior	251
				kol_otkazov_15prior	228

Рисунок 49 – Временные характеристики планировщика с локальными очередями задач с относительным обслуживанием в условиях $\rho = 0,71$

На рисунках 50–51 приведены результаты моделирования планировщика с локальными очередями задач с относительным обслуживанием для $\rho = 0,42$, $\rho = 0,29$.

sr_vr_reb_sist	12.6441	sr_vr_ojid_ocher	3.54491	kol_sozd_zayavok	572881
sr_vr_reb_sist_0prior	12.5213	sr_vr_ojid_ocher_0prior	3.38012	kol_otkazov	4
sr_vr_reb_sist_1prior	12.4516	sr_vr_ojid_ocher_1prior	3.34088	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	12.5337	sr_vr_ojid_ocher_2prior	3.39384	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	12.4693	sr_vr_ojid_ocher_3prior	3.40682	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	12.5476	sr_vr_ojid_ocher_4prior	3.47037	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	12.6739	sr_vr_ojid_ocher_5prior	3.50967	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	12.6703	sr_vr_ojid_ocher_6prior	3.50722	kol_otkazov_5prior	1
sr_vr_reb_sist_7prior	12.7184	sr_vr_ojid_ocher_7prior	3.52687	kol_otkazov_6prior	1
sr_vr_reb_sist_8prior	12.5541	sr_vr_ojid_ocher_8prior	3.53455	kol_otkazov_7prior	1
sr_vr_reb_sist_9prior	12.7116	sr_vr_ojid_ocher_9prior	3.59107	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	12.6671	sr_vr_ojid_ocher_10prior	3.59864	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	12.7181	sr_vr_ojid_ocher_11prior	3.61256	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	12.6627	sr_vr_ojid_ocher_12prior	3.60784	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	12.772	sr_vr_ojid_ocher_13prior	3.70974	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	12.825	sr_vr_ojid_ocher_14prior	3.74197	kol_otkazov_13prior	1
sr_vr_reb_sist_15prior	12.8088	sr_vr_ojid_ocher_15prior	3.78441	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 50 – Временные характеристики планировщика с локальными очередями с относительным обслуживанием в условиях $\rho = 0,42$

 sr_vr_reb_sist	11.2489	 sr_vr_ojid_ocher	2.16798	 kol_sozd_zayavok	393500
 sr_vr_reb_sist_0prior	11.3184	 sr_vr_ojid_ocher_0prior	2.13244	 kol_otkazov	0
 sr_vr_reb_sist_1prior	11.186	 sr_vr_ojid_ocher_1prior	2.15969	 kol_otkazov_0prior	0
 sr_vr_reb_sist_2prior	11.1447	 sr_vr_ojid_ocher_2prior	2.13139	 kol_otkazov_1prior	0
 sr_vr_reb_sist_3prior	11.2078	 sr_vr_ojid_ocher_3prior	2.16523	 kol_otkazov_2prior	0
 sr_vr_reb_sist_4prior	11.1907	 sr_vr_ojid_ocher_4prior	2.17057	 kol_otkazov_3prior	0
 sr_vr_reb_sist_5prior	11.1193	 sr_vr_ojid_ocher_5prior	2.11098	 kol_otkazov_4prior	0
 sr_vr_reb_sist_6prior	11.1613	 sr_vr_ojid_ocher_6prior	2.13941	 kol_otkazov_5prior	0
 sr_vr_reb_sist_7prior	11.2585	 sr_vr_ojid_ocher_7prior	2.17208	 kol_otkazov_6prior	0
 sr_vr_reb_sist_8prior	11.3905	 sr_vr_ojid_ocher_8prior	2.21685	 kol_otkazov_7prior	0
 sr_vr_reb_sist_9prior	11.2616	 sr_vr_ojid_ocher_9prior	2.15985	 kol_otkazov_8prior	0
 sr_vr_reb_sist_10prior	11.2979	 sr_vr_ojid_ocher_10prior	2.19	 kol_otkazov_9prior	0
 sr_vr_reb_sist_11prior	11.3701	 sr_vr_ojid_ocher_11prior	2.1945	 kol_otkazov_10prior	0
 sr_vr_reb_sist_12prior	11.3343	 sr_vr_ojid_ocher_12prior	2.19198	 kol_otkazov_11prior	0
 sr_vr_reb_sist_13prior	11.2631	 sr_vr_ojid_ocher_13prior	2.15815	 kol_otkazov_12prior	0
 sr_vr_reb_sist_14prior	11.2139	 sr_vr_ojid_ocher_14prior	2.22215	 kol_otkazov_13prior	0
 sr_vr_reb_sist_15prior	11.2623	 sr_vr_ojid_ocher_15prior	2.17174	 kol_otkazov_14prior	0
				 kol_otkazov_15prior	0

Рисунок 51 – Временные характеристики планировщика с локальными очередями задач с относительным обслуживанием в условиях $\rho = 0,29$

На рисунках 52–53 приведены результаты моделирования планировщика с локальными очередями задач с относительным обслуживанием для $\rho = 0,95$, $\rho = 0,71$.

 sr_vr_reb_sist	33.4612	 sr_vr_ojid_ocher	24.3341	 kol_sozd_zayavok	1269183
 sr_vr_reb_sist_0prior	33.5059	 sr_vr_ojid_ocher_0prior	24.3407	 kol_otkazov	82975
 sr_vr_reb_sist_1prior	33.4952	 sr_vr_ojid_ocher_1prior	24.3566	 kol_otkazov_0prior	5286
 sr_vr_reb_sist_2prior	33.4679	 sr_vr_ojid_ocher_2prior	24.3303	 kol_otkazov_1prior	5297
 sr_vr_reb_sist_3prior	33.2483	 sr_vr_ojid_ocher_3prior	24.2009	 kol_otkazov_2prior	5281
 sr_vr_reb_sist_4prior	33.6511	 sr_vr_ojid_ocher_4prior	24.4835	 kol_otkazov_3prior	5172
 sr_vr_reb_sist_5prior	33.4463	 sr_vr_ojid_ocher_5prior	24.3579	 kol_otkazov_4prior	5152
 sr_vr_reb_sist_6prior	33.4559	 sr_vr_ojid_ocher_6prior	24.3424	 kol_otkazov_5prior	5121
 sr_vr_reb_sist_7prior	33.428	 sr_vr_ojid_ocher_7prior	24.3037	 kol_otkazov_6prior	5098
 sr_vr_reb_sist_8prior	33.4009	 sr_vr_ojid_ocher_8prior	24.2844	 kol_otkazov_7prior	5196
 sr_vr_reb_sist_9prior	33.413	 sr_vr_ojid_ocher_9prior	24.294	 kol_otkazov_8prior	5247
 sr_vr_reb_sist_10prior	33.5349	 sr_vr_ojid_ocher_10prior	24.294	 kol_otkazov_9prior	5221
 sr_vr_reb_sist_11prior	33.5349	 sr_vr_ojid_ocher_11prior	24.3871	 kol_otkazov_10prior	5315
 sr_vr_reb_sist_12prior	33.2698	 sr_vr_ojid_ocher_12prior	24.177	 kol_otkazov_11prior	5182
 sr_vr_reb_sist_13prior	33.5051	 sr_vr_ojid_ocher_13prior	24.3238	 kol_otkazov_12prior	5124
 sr_vr_reb_sist_14prior	33.4994	 sr_vr_ojid_ocher_14prior	24.3443	 kol_otkazov_13prior	5089
 sr_vr_reb_sist_15prior	33.4855	 sr_vr_ojid_ocher_15prior	24.3465	 kol_otkazov_14prior	5050
	33.5694	 sr_vr_ojid_ocher_15prior	24.4701	 kol_otkazov_15prior	5144

Рисунок 52 – Временные характеристики планировщика с локальными очередями задач с беспriorитетным обслуживанием в условиях $\rho = 0,95$

sr_vr_reb_sist	20.2986	sr_vr_ojid_ocher	11.1787	kol_sozd_zayavok	962019
sr_vr_reb_sist_0prior	20.2534	sr_vr_ojid_ocher_0prior	11.1369	kol_otkazov	3764
sr_vr_reb_sist_1prior	20.3779	sr_vr_ojid_ocher_1prior	11.2871	kol_otkazov_0prior	239
sr_vr_reb_sist_2prior	20.36	sr_vr_ojid_ocher_2prior	11.2168	kol_otkazov_1prior	249
sr_vr_reb_sist_3prior	20.2303	sr_vr_ojid_ocher_3prior	11.0773	kol_otkazov_2prior	236
sr_vr_reb_sist_4prior	20.2007	sr_vr_ojid_ocher_4prior	11.0924	kol_otkazov_3prior	230
sr_vr_reb_sist_5prior	20.255	sr_vr_ojid_ocher_5prior	11.1313	kol_otkazov_4prior	235
sr_vr_reb_sist_6prior	20.3012	sr_vr_ojid_ocher_6prior	11.1739	kol_otkazov_5prior	244
sr_vr_reb_sist_7prior	20.214	sr_vr_ojid_ocher_7prior	11.1135	kol_otkazov_6prior	207
sr_vr_reb_sist_8prior	20.3148	sr_vr_ojid_ocher_8prior	11.1982	kol_otkazov_7prior	271
sr_vr_reb_sist_9prior	20.4064	sr_vr_ojid_ocher_9prior	11.2774	kol_otkazov_8prior	239
sr_vr_reb_sist_10prior	20.2533	sr_vr_ojid_ocher_10prior	11.1511	kol_otkazov_9prior	213
sr_vr_reb_sist_11prior	20.353	sr_vr_ojid_ocher_11prior	11.2877	kol_otkazov_10prior	243
sr_vr_reb_sist_12prior	20.331	sr_vr_ojid_ocher_12prior	11.1643	kol_otkazov_11prior	249
sr_vr_reb_sist_13prior	20.2822	sr_vr_ojid_ocher_13prior	11.1093	kol_otkazov_12prior	225
sr_vr_reb_sist_14prior	20.3334	sr_vr_ojid_ocher_14prior	11.2353	kol_otkazov_13prior	260
sr_vr_reb_sist_15prior	20.3125	sr_vr_ojid_ocher_15prior	11.2084	kol_otkazov_14prior	220
				kol_otkazov_15prior	204

Рисунок 53 – Временные характеристики планировщика с локальными очередями задач с бесприоритетным обслуживанием в условиях $\rho = 0,71$

На рисунках 54–55 приведены результаты моделирования планировщика с локальными очередями задач с относительным обслуживанием для $\rho = 0,42$, $\rho = 0,29$.

sr_vr_reb_sist	12.6833	sr_vr_ojid_ocher	3.56443	kol_sozd_zayavok	572881
sr_vr_reb_sist_0prior	12.6911	sr_vr_ojid_ocher_0prior	3.54929	kol_otkazov	1
sr_vr_reb_sist_1prior	12.6101	sr_vr_ojid_ocher_1prior	3.53098	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	12.6795	sr_vr_ojid_ocher_2prior	3.55481	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	12.6763	sr_vr_ojid_ocher_3prior	3.55058	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	12.617	sr_vr_ojid_ocher_4prior	3.53482	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	12.7428	sr_vr_ojid_ocher_5prior	3.5993	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	12.6365	sr_vr_ojid_ocher_6prior	3.56126	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	12.7357	sr_vr_ojid_ocher_7prior	3.57542	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	12.789	sr_vr_ojid_ocher_8prior	3.56761	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	12.6586	sr_vr_ojid_ocher_9prior	3.52673	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	12.6627	sr_vr_ojid_ocher_10prior	3.57847	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	12.7342	sr_vr_ojid_ocher_11prior	3.60989	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	12.691	sr_vr_ojid_ocher_12prior	3.55377	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	12.6771	sr_vr_ojid_ocher_13prior	3.56918	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	12.6506	sr_vr_ojid_ocher_14prior	3.57069	kol_otkazov_13prior	1
sr_vr_reb_sist_15prior	12.6801	sr_vr_ojid_ocher_15prior	3.59779	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 54 – Временные характеристики планировщика с локальными очередями с бесприоритетным обслуживанием в условиях $\rho = 0,42$

sr_vr_reb_sist	11.2961	sr_vr_ojid_ocher	2.17063	kol_sozd_zayavok	393500
sr_vr_reb_sist_0prior	11.3426	sr_vr_ojid_ocher_0prior	2.1478	kol_otkazov	0
sr_vr_reb_sist_1prior	11.2843	sr_vr_ojid_ocher_1prior	2.19169	kol_otkazov_0prior	0
sr_vr_reb_sist_2prior	11.2748	sr_vr_ojid_ocher_2prior	2.14469	kol_otkazov_1prior	0
sr_vr_reb_sist_3prior	11.3964	sr_vr_ojid_ocher_3prior	2.17154	kol_otkazov_2prior	0
sr_vr_reb_sist_4prior	11.3466	sr_vr_ojid_ocher_4prior	2.17905	kol_otkazov_3prior	0
sr_vr_reb_sist_5prior	11.3004	sr_vr_ojid_ocher_5prior	2.15868	kol_otkazov_4prior	0
sr_vr_reb_sist_6prior	11.3339	sr_vr_ojid_ocher_6prior	2.16844	kol_otkazov_5prior	0
sr_vr_reb_sist_7prior	11.2323	sr_vr_ojid_ocher_7prior	2.19493	kol_otkazov_6prior	0
sr_vr_reb_sist_8prior	11.3091	sr_vr_ojid_ocher_8prior	2.15644	kol_otkazov_7prior	0
sr_vr_reb_sist_9prior	11.2041	sr_vr_ojid_ocher_9prior	2.18986	kol_otkazov_8prior	0
sr_vr_reb_sist_10prior	11.2492	sr_vr_ojid_ocher_10prior	2.18716	kol_otkazov_9prior	0
sr_vr_reb_sist_11prior	11.2696	sr_vr_ojid_ocher_11prior	2.19748	kol_otkazov_10prior	0
sr_vr_reb_sist_12prior	11.3845	sr_vr_ojid_ocher_12prior	2.15414	kol_otkazov_11prior	0
sr_vr_reb_sist_13prior	11.2825	sr_vr_ojid_ocher_13prior	2.15864	kol_otkazov_12prior	0
sr_vr_reb_sist_14prior	11.2862	sr_vr_ojid_ocher_14prior	2.18369	kol_otkazov_13prior	0
sr_vr_reb_sist_15prior	11.2409	sr_vr_ojid_ocher_15prior	2.14646	kol_otkazov_14prior	0
				kol_otkazov_15prior	0

Рисунок 55 – Временные характеристики планировщика с локальными очередями задач с беспriorитетным обслуживанием в условиях $\rho = 0,29$

3.6 Выработка критериев сравнительной оценки планировщиков

На основе результатов моделирования было выполнено сравнение основных характеристик разработанных планировщиков с глобальной и локальными очередями. Для сравнения рассматривались три режима обслуживания заявок: с абсолютными приоритетами, с относительными приоритетами и беспriorитетный режим.

В качестве основных критериев сравнительной оценки использовались: общее число созданных заявок, число потерянных заявок, число потерянных заявок по каждому приоритету, общее среднее время пребывания заявки в системе, а также среднее время пребывания заявок каждого приоритета. Такой набор характеристик позволяет оценить не только общую эффективность работы планировщика, но и влияние выбранного режима обслуживания на заявки разных приоритетов.

Результаты моделирования показали, что при высокой загрузке системы число потерянных заявок существенно возрастает для всех вариантов планировщиков. При уменьшении интенсивности входного потока количество

отказов снижается, а при низкой загрузке возможен режим работы без потерь. Например, для планировщика с глобальной очередью при нагрузке $\rho = 0,95$ наблюдаются потери заявок во всех трёх режимах обслуживания, а при нагрузке $\rho = 0,29$ число потерянных заявок становится равным нулю для абсолютного, относительного и бесприоритетного режимов.

Сравнение режимов обслуживания показало, что абсолютные приоритеты обеспечивают преимущество заявкам с более высоким приоритетом: для них среднее время пребывания в системе оказывается меньше, чем для заявок с низким приоритетом. Однако при высокой нагрузке низкоприоритетные заявки могут находиться в системе значительно дольше. В режиме с относительными приоритетами различия между приоритетами выражены слабее, так как уже выполняемая задача не вытесняется. В бесприоритетном режиме заявки обслуживаются без учёта приоритета, поэтому средние времена пребывания по приоритетам имеют более равномерный характер.

Планировщики с глобальной очередью обеспечивают более равномерное использование процессоров, так как каждый освободившийся процессор может выбрать следующую заявку из общей очереди. Планировщики с локальными очередями имеют более сложную структуру распределения заявок: поступившая заявка направляется в одну из локальных очередей, после чего дальнейший выбор выполняется в пределах этой очереди. Из-за этого возможна неравномерность заполнения локальных очередей, что может влиять на число отказов и среднее время пребывания заявок в системе.

Результаты измерения характеристик показаны в таблицах 4–7 соответственно.

Таблица 4 – Временные характеристики планировщика с глобальной очередью и планировщика с локальными очередями при нагрузке процессоров ($\rho = 0,95$)

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Созд. задачи, всего	1269185					
Потер. задачи, всего	95313	108905	110185	114049	82731	82975
Потер. задачи, 0 приор.	5931	6752	6925	7101	5184	5286
Потер. задачи, 1 приор.	5906	6795	6864	7190	5233	5297
Потер. задачи, 2 приор.	6038	6959	7015	7121	5116	5281
Потер. задачи, 3 приор.	5973	6737	6887	7149	5245	5172
Потер. задачи, 4 приор.	5995	6800	6990	7093	5231	5152
Потер. задачи, 5 приор.	5969	6711	7020	7154	5105	5121
Потер. задачи, 6 приор.	5990	6844	6913	6985	5060	5098
Потер. задачи, 7 приор.	5949	6732	6775	7159	5219	5196
Потер. задачи, 8 приор.	5870	6855	6941	7087	5267	5247
Потер. задачи, 9 приор.	6003	6736	6784	7166	5114	5221
Потер. задачи, 10 приор.	6040	6853	6899	7301	5057	5315
Потер. задачи, 11 приор.	5986	6796	6686	7106	5085	5182
Потер. задачи, 12 приор.	5938	6804	6736	7193	5214	5124
Потер. задачи, 13 приор.	5889	6893	6878	6999	5261	5089
Потер. задачи, 14 приор.	5824	6774	6874	7130	5086	5050
Потер. задачи, 15 приор.	6012	6864	6948	7115	5254	5144
Общее ср. вр. вып.	35.298	33.737	32.5186	35.3498	33.197	33.4612
Ср. вр. вып., 0 приор.	9.0592	11.9987	32.391	10.498	18.066	33.5059
Ср. вр. вып., 1 приор.	9.1154	12.951	32.542	11.142	19.149	33.4952
Ср. вр. вып., 2 приор.	9.1443	12.5855	32.5272	11.9972	19.8585	33.4679
Ср. вр. вып., 3 приор.	9.1703	12.9181	32.5498	12.7151	20.5667	33.2483
Ср. вр. вып., 4 приор.	9.2057	13.778	32.479	13.491	21.4796	33.6511

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. вып., 5 приор.	9.3516	14.1075	32.493	14.9338	22.5265	33.4463
Ср. вр. вып., 6 приор.	9.6093	14.719	32.529	16.302	23.6336	33.459
Ср. вр. вып., 7 приор.	10.0136	15.954	32.4915	17.952	24.8854	33.428
Ср. вр. вып., 8 приор.	10.789	17.3178	32.492	19.9993	26.7435	33.4009
Ср. вр. вып., 9 приор.	12.172	19.2	32.631	22.666	28.6938	33.413
Ср. вр. вып., 10 приор.	14.2594	21.797	32.634	26.143	31.1507	33.5349
Ср. вр. вып., 11 приор.	17.906	26.2509	32.5966	30.892	34.572	33.2698
Ср. вр. вып., 12 приор.	24.624	32.6727	32.413	38.2163	39.228	33.5051
Ср. вр. вып., 13 приор.	38.3989	45.1591	32.5728	51.017	46.7008	33.4994
Ср. вр. вып., 14 приор.	73.466	73.3485	32.4325	77.3403	60.124	33.4855
Ср. вр. вып., 15 приор.	294.23	195.664	32.541	187.898	93.4215	33.5694
Общее ср. вр. отклика.	27.105	23.853	23.4961	26.3451	24.102	24.3341
Ср. вр. отклика, 0 приор.	1.0003	2.25374	23.3942	1.36754	9.45025	24.307
Ср. вр. отклика, 1 приор.	1.003	2.49673	23.5346	2.05376	10.02	24.3566
Ср. вр. отклика, 2 приор.	1.01481	2.79166	23.4955	2.842	10.7298	24.3303
Ср. вр. отклика, 3 приор.	1.04524	3.16856	23.4861	3.7193	11.541	24.009
Ср. вр. отклика, 4 приор.	1.12981	3.63839	23.4637	4.6778	12.3801	24.835
Ср. вр. отклика, 5 приор.	1.29511	4.25879	23.5282	5.9567	13.4386	24.3579
Ср. вр. отклика, 6 приор.	1.56679	4.97243	23.4858	7.36146	14.536	24.3424
Ср. вр. отклика, 7 приор.	2.0365	6.02265	23.489	8.82284	15.7843	24.037
Ср. вр. отклика, 8 приор.	2.8281	7.42058	23.432	10.953	17.6335	24.2844
Ср. вр. отклика, 9 приор.	4.1646	9.29792	23.55	13.6603	19.6095	24.29
Ср. вр. отклика, 10 приор.	6.24208	11.69	23.5453	17.612	22.1079	24.3871
Ср. вр. отклика, 11 приор.	9.79413	16.2963	23.5935	21.9541	25.3614	24.17
Ср. вр. отклика, 12 приор.	16.39	22.6731	23.436	29.338	30.159	24.238
Ср. вр. отклика, 13 приор.	29.9598	35.138	23.5296	42.951	37.598	24.343
Ср. вр. отклика, 14 приор.	64.7498	63.39	23.4277	68.4225	50.8485	24.3465

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. отклика, 15 приор.	285.202	185.608	23.5435	178.796	84.264	24.4701

По таблице 4 видно, что при нагрузке $\rho = 0,95$ система работает в условиях высокой загруженности процессоров. Из-за этого увеличивается время ожидания задач в очередях, возрастает количество потерянных задач, а также увеличиваются среднее время отклика и среднее время выполнения задачи. Это говорит о том, что при такой нагрузке планировщики работают близко к предельному режиму.

При сравнении моделей по количеству потерянных задач, среднему времени отклика и среднему времени выполнения задачи лучшие результаты показывают модели планировщиков задач с глобальной очередью по среднему времени отклика и среднему времени выполнения задачи, а по количеству потерянных задач лучший результат показывает организация с локальными очередями. Это означает, что модели планировщиков задач с глобальной очередью лучше распределяют задачи между процессорами и уменьшают время ожидания перед началом обработки, а модели планировщиков задач с локальными очередями при высокой нагрузке эффективнее снижают вероятность потери заявок.

При сравнении типов обслуживания задач можно отметить, что бесприоритетный тип обслуживания обеспечивает наиболее равномерные значения среднего времени отклика и среднего времени выполнения задачи по приоритетам, но практически не учитывает важность заявок. Абсолютный тип обслуживания сильнее выделяет высокоприоритетные задачи, однако ухудшает характеристики низкоприоритетных заявок. Относительный тип обслуживания является более сбалансированным вариантом, так как учитывает приоритеты без резкого ухудшения обслуживания задач с низким приоритетом.

На рисунке 56 представлена диаграмма среднего времени отклика по приоритетам для моделей планировщиков при нагрузке $\rho = 0,95$.

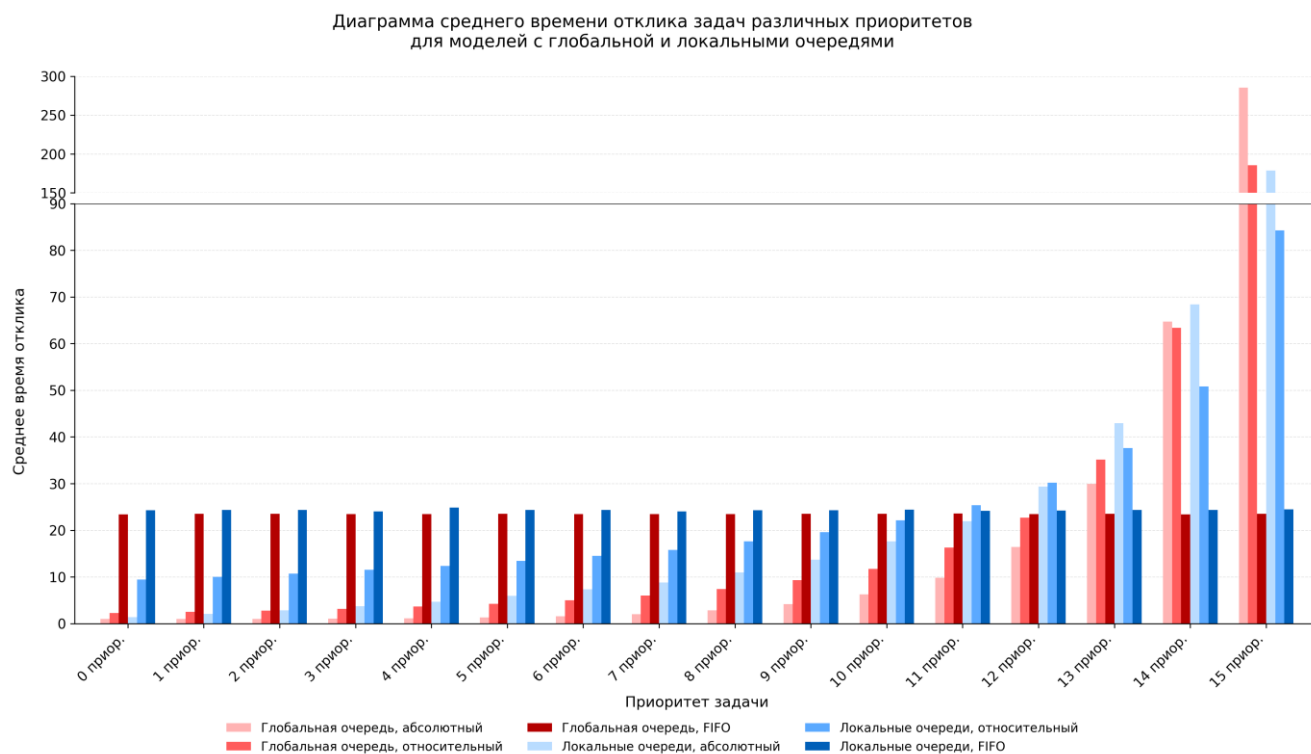


Рисунок 56 – Диаграмма времени отклика по приоритетам для всех моделей при $p = 0,95$

Таблица 5 – Временные характеристики планировщика с глобальной очередью и планировщика с локальными очередями при нагрузке процессоров ($p = 0,71$)

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Созд. задачи, всего	962019					
Потер. задачи, всего	4120	3194	3590	6722	3785	3754
Потер. задачи, 0 приор.	276	213	229	388	237	239
Потер. задачи, 1 приор.	263	212	225	416	216	249
Потер. задачи, 2 приор.	244	212	226	436	252	236
Потер. задачи, 3 приор.	284	207	234	414	245	230
Потер. задачи, 4 приор.	261	184	200	453	257	235
Потер. задачи, 5 приор.	224	197	214	429	237	244
Потер. задачи, 6 приор.	244	196	221	383	210	207

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Потер. задачи, 7 приор.	251	198	220	418	241	271
Потер. задачи, 8 приор.	256	189	223	403	226	239
Потер. задачи, 9 приор.	254	203	261	433	233	213
Потер. задачи, 10 приор.	244	199	201	382	263	243
Потер. задачи, 11 приор.	241	180	238	398	236	249
Потер. задачи, 12 приор..	291	200	217	463	224	225
Потер. задачи, 13 приор.	255	197	232	414	229	260
Потер. задачи, 14 приор.	264	188	215	433	251	220
Потер. задачи, 15 приор.	268	219	234	459	228	204
Общее ср. вр. вып.	17.1828	16.8241	15.944	21.014	20.1504	20.2986
Ср. вр. вып., 0 приор.	9.10625	11.35	16.026	10.4147	16.072	20.2534
Ср. вр. вып., 1 приор.	9.09739	11.5563	15.9987	10.8715	16.395	20.3779
Ср. вр. вып., 2 приор.	9.03411	11.7472	15.9609	11.5872	16.6179	20.36
Ср. вр. вып., 3 приор.	9.1562	11.9185	15.95	12.3226	16.943	20.203
Ср. вр. вып., 4 приор.	9.2093	12.1252	15.838	13.0971	17.321	20.2007
Ср. вр. вып., 5 приор.	9.38068	12.5712	15.9821	13.835	17.862	20.255
Ср. вр. вып., 6 приор.	9.6532	13.0032	15.9247	15.0049	18.223	20.3012
Ср. вр. вып., 7 приор.	10.037	13.6688	15.8676	16.1753	18.679	20.214
Ср. вр. вып., 8 приор.	10.719	14.1099	15.9098	17.7028	19.2555	20.3148
Ср. вр. вып., 9 приор.	11.7656	14.9	16.033	19.3921	19.9815	20.4064
Ср. вр. вып., 10 приор.	13.021	16.0476	15.9782	21.453	20.8525	20.2533
Ср. вр. вып., 11 приор.	15.7923	17.7036	15.9643	24.161	21.6702	20.353
Ср. вр. вып., 12 приор.	19.9582	19.7064	15.9647	27.6308	22.6194	20.331
Ср. вр. вып., 13 приор.	26.3759	23.0141	15.8655	32.6319	24.399	20.282
Ср. вр. вып., 14 приор.	38.7119	28.2659	15.8687	40.858	26.1164	20.3334
Ср. вр. вып., 15 приор.	63.441	37.5903	15.9726	55.3714	29.328	20.3125
Общее ср. вр. отклика	8.92532	7.21521	7.2754	12.321	11.0616	11.1787

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. отклика, 0 приор.	1.00003	1.82596	7.29593	1.29316	6.96204	11.136
Ср. вр. отклика, 1 приор.	1.00258	1.96993	7.27456	1.85718	7.28372	11.2871
Ср. вр. отклика, 2 приор.	1.0124	2.16025	7.26243	2.4076	7.50339	11.216
Ср. вр. отклика, 3 приор.	1.0476	2.36546	7.28729	3.2418	7.88239	11.077
Ср. вр. отклика, 4 приор.	1.11762	2.61983	7.23762	3.96982	8.27173	11.0924
Ср. вр. отклика, 5 приор.	1.247	2.98102	7.30282	4.8623	8.76636	11.131
Ср. вр. отклика, 6 приор.	1.52	3.37103	7.2589	5.93782	9.12076	11.173
Ср. вр. отклика, 7 приор.	1.91654	3.87178	7.18122	7.09315	9.6153	11.113
Ср. вр. отклика, 8 приор.	2.57687	4.55062	7.21701	8.59302	10.1719	11.1984
Ср. вр. отклика, 9 приор.	3.55894	5.3456	7.33112	10.2705	10.9453	11.277
Ср. вр. отклика, 10 приор.	5.11221	6.4278	7.29061	12.4072	11.7281	11.151
Ср. вр. отклика, 11 приор.	7.46672	7.99699	7.31422	15.0411	12.575	11.287
Ср. вр. отклика, 12 приор.	11.5026	10.0851	7.30876	18.556	13.5535	11.164
Ср. вр. отклика, 13 приор.	17.8742	13.313	7.2457	23.5979	15.2977	11.109
Ср. вр. отклика, 14 приор.	30.05	18.508	7.22118	31.422	17.0703	11.235
Ср. вр. отклика, 15 приор.	54.5263	27.8381	7.32252	46.2606	20.1728	11.208

По таблице 5 видно, что при нагрузке $p = 0,71$ система работает стабильнее, чем при $p = 0,95$. Нагрузка на процессоры ниже, поэтому уменьшается количество потерянных задач, а также сокращаются среднее время отклика и среднее время выполнения задачи.

При сравнении моделей по количеству потерянных задач, среднему времени отклика и среднему времени выполнения задачи лучшие результаты показывают модели планировщиков задач с глобальной очередью, особенно модель планировщика задач с глобальной очередью с относительным типом обслуживания задач по количеству потерянных задач. Такая организация очереди позволяет равномернее распределять задачи между процессорами, снижать время ожидания

заявок перед началом обработки и уменьшать общее время нахождения задач в системе.

При сравнении типов обслуживания задач можно отметить, что бесприоритетный тип обслуживания обеспечивает наиболее равномерные значения среднего времени отклика и среднего времени выполнения задачи по приоритетам, но практически не учитывает важность заявок. Абсолютный тип обслуживания сильнее выделяет высокоприоритетные задачи, однако ухудшает характеристики низкоприоритетных заявок. Относительный тип обслуживания является более сбалансированным вариантом, так как учитывает приоритеты без резкого ухудшения обслуживания задач с низким приоритетом.

На рисунке 57 представлена диаграмма среднего времени отклика по приоритетам для моделей планировщиков при нагрузке $\rho = 0,71$.

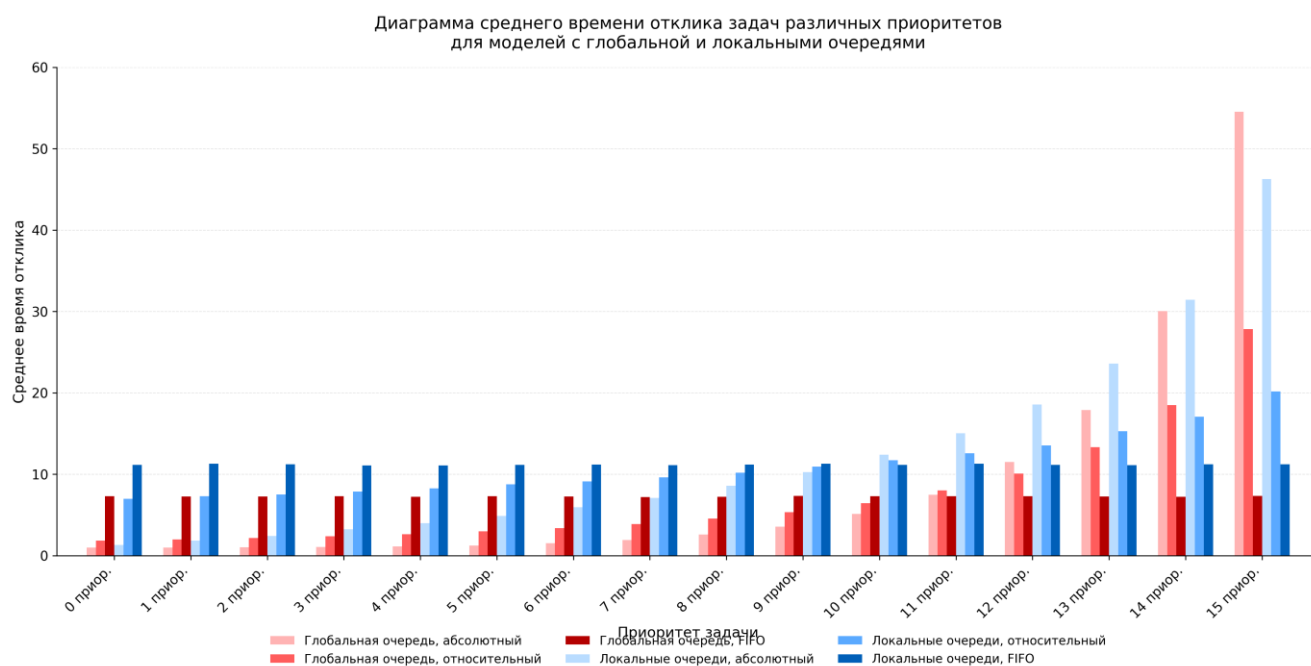


Рисунок 57 – Диаграмма времени отклика по приоритетам для всех моделей при $\rho = 0,71$

Таблица 6 – Временные характеристики планировщика с глобальной очередью и планировщика с локальными очередями при нагрузке процессоров ($\rho = 0,42$)

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Созд. задачи, всего	572881					
Потер. задачи, всего	0	0	0	0	0	0
Потер. задачи, 0 приор.	0	0	0	0	0	0
Потер. задачи, 1 приор.	0	0	0	0	0	0
Потер. задачи, 2 приор.	0	0	0	0	0	0
Потер. задачи, 3 приор.	0	0	0	0	0	0
Потер. задачи, 4 приор.	0	0	0	0	0	0
Потер. задачи, 5 приор.	0	0	0	0	0	0
Потер. задачи, 6 приор.	0	0	0	0	0	0
Потер. задачи, 7 приор.	0	0	0	0	0	0
Потер. задачи, 8 приор.	0	0	0	0	0	0
Потер. задачи, 9 приор.	0	0	0	0	0	0
Потер. задачи, 10 приор.	0	0	0	0	0	0
Потер. задачи, 11 приор.	0	0	0	0	0	0
Потер. задачи, 12 приор..	0	0	0	0	0	0
Потер. задачи, 13 приор.	0	0	0	1	1	1
Потер. задачи, 14 приор.	0	0	0	0	0	0
Потер. задачи, 15 приор.	0	0	0	0	0	0
Общее ср. вр. вып.	9.76684	10.8638	9.87454	12.527	12.6441	12.6833
Ср. вр. вып., 0 приор.	9.13857	10.4738	9.84602	10.1669	12.5213	12.6911
Ср. вр. вып., 1 приор.	9.04103	10.3984	9.92155	10.3761	12.4516	12.611
Ср. вр. вып., 2 приор.	9.11891	10.4478	9.83823	10.5855	12.5337	12.6795
Ср. вр. вып., 3 приор.	9.1345	10.5577	9.90854	10.9176	12.4693	12.6763
Ср. вр. вып., 4 приор.	9.11978	10.5506	9.89321	11.2304	12.5476	12.617

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. вып., 5 приор.	9.11155	10.6916	9.83611	11.51	12.6739	12.7248
Ср. вр. вып., 6 приор.	9.1781	10.7826	9.8907	11.7241	12.6703	12.6365
Ср. вр. вып., 7 приор.	9.25016	10.7293	9.85704	12.0728	12.7184	12.7357
Ср. вр. вып., 8 приор.	9.34287	10.9041	9.828	12.5406	12.5541	12.789
Ср. вр. вып., 9 приор.	9.49135	10.9197	9.93075	13.0373	12.7116	12.6568
Ср. вр. вып., 10 приор.	9.65579	10.88	9.86748	13.3476	12.6671	12.6027
Ср. вр. вып., 11 приор.	10.0473	11.1275	9.8913	13.8115	12.7181	12.7342
Ср. вр. вып., 12 приор.	10.2113	11.205	9.85217	14.2936	12.6627	12.691
Ср. вр. вып., 13 приор.	10.7616	11.2144	9.84222	14.8743	12.772	12.6771
Ср. вр. вып., 14 приор.	11.393	11.42	9.95665	15.6443	12.825	12.6506
Ср. вр. вып., 15 приор.	12.2604	11.59	9.83359	16.2766	12.808	12.6801
Общее ср. вр. отклика	1.62641	1.61577	1.62671	3.5537	3.54491	3.56443
Ср. вр. отклика, 0 приор.	1	1.22656	1.62693	1.09607	3.38012	3.54929
Ср. вр. отклика, 1 приор.	1.0006	1.25411	1.63785	1.31785	3.3408	3.53098
Ср. вр. отклика, 2 приор.	1.00179	1.280	1.63471	1.56263	3.39384	3.55481
Ср. вр. отклика, 3 приор.	1.00762	1.30777	1.61565	1.85689	3.4068	3.55058
Ср. вр. отклика, 4 приор.	1.0204	1.34431	1.66219	2.14603	3.47037	3.53482
Ср. вр. отклика, 5 приор.	1.04961	1.38594	1.61043	2.39819	3.50967	3.5993
Ср. вр. отклика, 6 приор.	1.09424	1.44771	1.60747	2.67344	3.50722	3.56126
Ср. вр. отклика, 7 приор.	1.15521	1.49297	1.6453	2.99821	3.52687	3.57542
Ср. вр. отклика, 8 приор.	1.25054	1.54089	1.63531	3.40553	3.53455	3.56761
Ср. вр. отклика, 9 приор.	1.3726	1.64215	1.62497	3.85912	3.59107	3.52673
Ср. вр. отклика, 10 приор.	1.53182	1.70153	1.61952	4.25471	3.59864	3.57847
Ср. вр. отклика, 11 приор.	1.79756	1.8002	1.63875	4.73268	3.61256	3.60989
Ср. вр. отклика, 12 приор.	2.0717	1.89316	1.62513	5.15674	3.60784	3.55377
Ср. вр. отклика, 13 приор.	2.54551	1.98262	1.60862	5.80871	3.70974	3.56918
Ср. вр. отклика, 14 приор.	3.1381	2.1899	1.631	6.46294	3.74197	3.57069

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. отклика, 15 приор.	3.97381	2.35724	1.60309	7.00726	3.78441	3.59779

По таблице 6 видно, что при нагрузке $\rho = 0,42$ система работает значительно стабильнее, чем при более высоких нагрузках. Нагрузка на процессоры снизилась, поэтому показатели работы планировщиков улучшились: уменьшилось количество потерянных задач, а также сократились среднее время отклика и среднее время выполнения задачи.

При сравнении моделей по количеству потерянных задач, среднему времени отклика и среднему времени выполнения задачи лучшие результаты показывают модели планировщиков задач с глобальной очередью. Такая организация очереди позволяет равномернее распределять задачи между процессорами, снижать время ожидания заявок перед началом обработки и уменьшать общее время нахождения задач в системе.

При сравнении типов обслуживания задач можно отметить, что беспriorитетный тип обслуживания обеспечивает наиболее равномерные значения среднего времени отклика и среднего времени выполнения задачи по приоритетам, но практически не учитывает важность заявок. Абсолютный тип обслуживания выделяет высокоприоритетные задачи, уменьшая их время отклика, однако увеличивает ожидание задач с низким приоритетом. Относительный тип обслуживания является более сбалансированным вариантом, так как учитывает приоритеты без резкого ухудшения характеристик низкоприоритетных задач.

На рисунке 58 представлена диаграмма среднего времени отклика по приоритетам для моделей планировщиков при нагрузке $\rho = 0,42$.

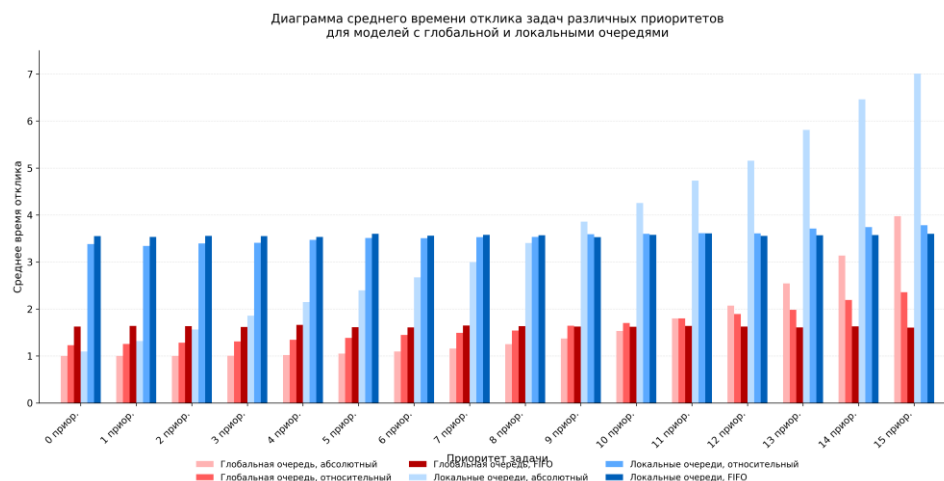


Рисунок 58 – Диаграмма времени отклика по приоритетам для всех моделей при $p = 0,42$

Таблица 7 – Временные характеристики планировщика с глобальной очередью и планировщика с локальными очередями при нагрузке процессоров ($p = 0,29$)

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Созд. задачи, всего	393500					
Потер. задачи, всего	0	0	0	0	0	0
Потер. задачи, 0 приор.	0	0	0	0	0	0
Потер. задачи, 1 приор.	0	0	0	0	0	0
Потер. задачи, 2 приор.	0	0	0	0	0	0
Потер. задачи, 3 приор.	0	0	0	0	0	0
Потер. задачи, 4 приор.	0	0	0	0	0	0
Потер. задачи, 5 приор.	0	0	0	0	0	0
Потер. задачи, 6 приор.	0	0	0	0	0	0
Потер. задачи, 7 приор.	0	0	0	0	0	0
Потер. задачи, 8 приор.	0	0	0	0	0	0
Потер. задачи, 9 приор.	0	0	0	0	0	0
Потер. задачи, 10 приор.	0	0	0	0	0	0

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Потер. задачи, 11 приор.	0	0	0	0	0	0
Потер. задачи, 12 приор.	0	0	0	0	0	0
Потер. задачи, 13 приор.	0	0	0	0	0	0
Потер. задачи, 14 приор.	0	0	0	0	0	0
Потер. задачи, 15 приор.	0	0	0	0	0	0
Общее ср. вр. вып.	9.24695	10.311	9.2479	11.2141	11.2489	11.2961
Ср. вр. вып., 0 приор.	9.12544	10.3284	9.3261	10.0642	11.3184	11.3426
Ср. вр. вып., 1 приор.	9.01491	10.3125	9.27065	10.279	11.186	11.2843
Ср. вр. вып., 2 приор.	9.06604	10.3202	9.30049	10.3596	11.447	11.2748
Ср. вр. вып., 3 приор.	9.15094	10.2707	9.36991	10.4875	11.2078	11.3964
Ср. вр. вып., 4 приор.	9.06091	10.2718	9.20224	10.7574	11.1907	11.3466
Ср. вр. вып., 5 приор.	9.15884	10.2072	9.26804	10.8786	11.193	11.3004
Ср. вр. вып., 6 приор.	9.0528	10.2835	9.2938	10.8722	11.613	11.3339
Ср. вр. вып., 7 приор.	9.00217	10.323	9.32721	11.077	11.2585	11.223
Ср. вр. вып., 8 приор.	9.15809	10.3065	9.31635	11.1827	11.3905	11.3091
Ср. вр. вып., 9 приор.	9.13964	10.2606	9.15886	11.3328	11.2616	11.2041
Ср. вр. вып., 10 приор.	9.24555	10.2964	9.30791	11.7	11.2979	11.2492
Ср. вр. вып., 11 приор.	9.26111	10.3391	9.31568	11.6972	11.3701	11.2696
Ср. вр. вып., 12 приор.	9.40439	10.35	9.2697	11.9267	11.3343	11.3845
Ср. вр. вып., 13 приор.	9.61784	10.384	9.3518	12.0422	11.2631	11.2825
Ср. вр. вып., 14 приор.	9.65626	10.3546	9.25796	12.1616	11.2139	11.2862
Ср. вр. вып., 15 приор.	9.76039	10.3676	9.3461	12.5789	11.2623	11.2409
Общее ср. вр. отклика	1.1528	1.1503	1.6387	2.13289	2.16798	2.17063
Ср. вр. отклика, 0 приор.	1	1.08264	1.16641	1.05779	2.13244	2.1478
Ср. вр. отклика, 1 приор.	1.00008	1.0844	1.16877	1.18512	2.15969	2.19169
Ср. вр. отклика, 2 приор.	1.00128	1.0854	1.15779	1.27105	2.1313	2.14469
Ср. вр. отклика, 3 приор.	1.00195	1.0976	1.17162	1.41649	2.16523	2.17154

Характеристика	Глобальная очередь			Локальные очереди		
	Абсолют.	Относит.	FIFO	Абсолют.	Относит.	FIFO
Ср. вр. отклика, 4 приор.	1.00505	1.10053	1.16976	1.56696	2.17057	2.17905
Ср. вр. отклика, 5 приор.	1.01279	1.11782	1.16228	1.72346	2.11098	2.15868
Ср. вр. отклика, 6 приор.	1.0265	1.11779	1.16133	1.87729	2.1394	2.16844
Ср. вр. отклика, 7 приор.	1.03517	1.13638	1.16846	1.98678	2.17208	2.19493
Ср. вр. отклика, 8 приор.	1.06469	1.1524	1.17114	2.1268	2.2168	2.15644
Ср. вр. отклика, 9 приор.	1.08464	1.16524	1.16222	2.30319	2.1598	2.18986
Ср. вр. отклика, 10 приор.	1.13955	1.18437	1.16305	2.48701	2.19	2.18716
Ср. вр. отклика, 11 приор.	1.19715	1.1924	1.16558	2.63346	2.1945	2.19748
Ср. вр. отклика, 12 приор.	1.2758	1.19718	1.15637	2.86783	2.1918	2.15414
Ср. вр. отклика, 13 приор.	1.37643	1.21695	1.14337	2.97058	2.15815	2.15864
Ср. вр. отклика, 14 приор.	1.52075	1.23435	1.16583	3.157	2.2215	2.18369
Ср. вр. отклика, 15 приор.	1.69611	1.24511	1.1681	3.47122	2.17174	2.14646

По таблице 7 видно, что при нагрузке $\rho = 0,29$ система работает в наиболее стабильном режиме. Нагрузка на процессоры низкая, поэтому все модели планировщиков справляются с поступающим потоком задач без перегрузки, а количество потерянных задач отсутствует.

При сравнении моделей по количеству потерянных задач, среднему времени отклика и среднему времени выполнения задачи различия между моделями становятся минимальными. Это связано с тем, что ресурсов системы достаточно для обработки всех поступающих задач, очереди не переполняются, а задачи быстро поступают на обработку. При этом модели планировщиков задач с глобальной очередью сохраняют хорошие показатели по среднему времени отклика и среднему времени выполнения задачи за счет равномерного распределения заявок между процессорами.

При сравнении типов обслуживания задач можно отметить, что бесприоритетный тип обслуживания обеспечивает наиболее равномерные значения среднего времени отклика и среднего времени выполнения задачи по

приоритетам, но практически не учитывает важность заявок. Абсолютный и относительный типы обслуживания сохраняют учет приоритетов, однако при низкой нагрузке разница между высокоприоритетными и низкоприоритетными задачами выражена слабо.

На рисунке 59 представлена диаграмма среднего времени отклика по приоритетам для моделей планировщиков при нагрузке $p = 0,29$.

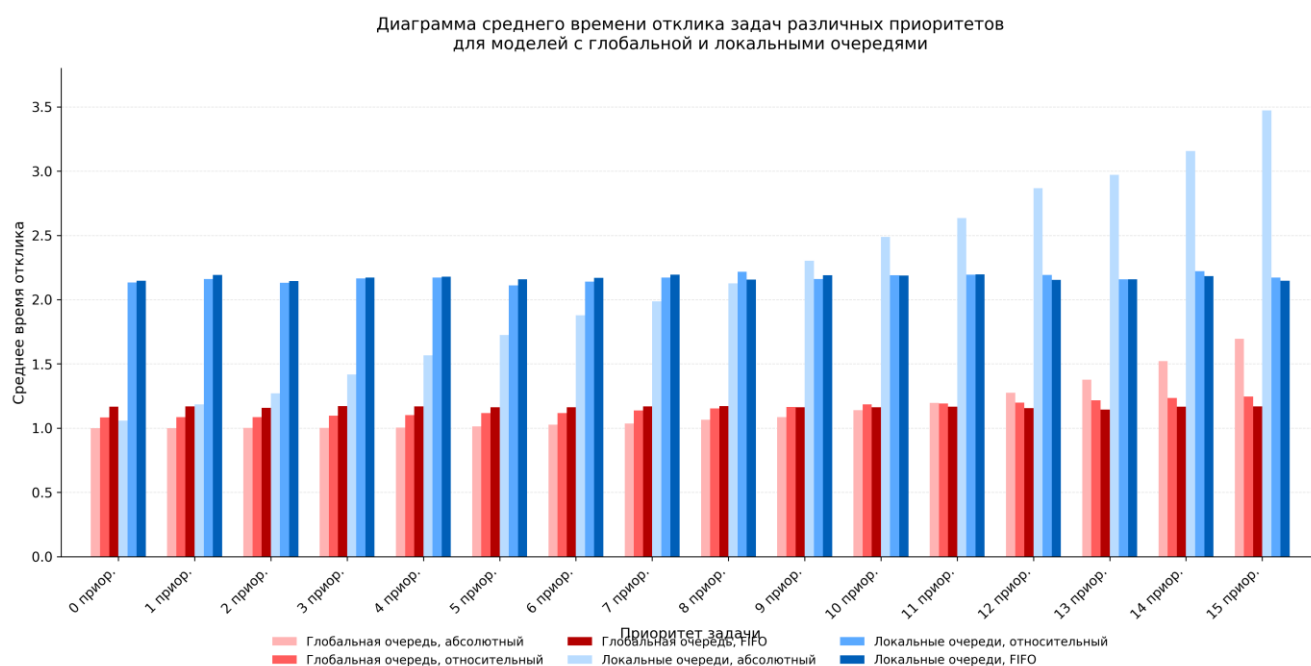


Рисунок 59 – Диаграмма времени отклика по приоритетам для всех моделей при $p = 0,29$

При сравнении организации очередей видно, что модели планировщиков задач с глобальной очередью в большинстве случаев показывают лучшие результаты по среднему времени отклика и среднему времени выполнения задачи. Например, при $p = 0,71$ среднее время отклика у моделей с глобальной очередью находится в пределах от 7,21521 до 8,92532, а у моделей с локальными очередями – от 11,0616 до 12,321. Среднее время выполнения задачи при $p = 0,71$ у моделей с глобальной очередью составляет от 15,944 до 17,1828, а у моделей с локальными очередями – от 20,1504 до 21,014. Это показывает, что глобальная очередь лучше распределяет задачи между процессорами и уменьшает время ожидания.

По количеству потерянных задач результат зависит от нагрузки. При $\rho = 0,95$ меньше всего потерь показывает модель планировщика задач с локальными очередями с относительным типом обслуживания задач – 82731 потерянная задача. Однако при $\rho = 0,71$ лучший результат уже у модели планировщика задач с глобальной очередью с относительным типом обслуживания задач – 3194 потерянные задачи. При $\rho = 0,42$ потери у моделей с глобальной очередью отсутствуют, а при $\rho = 0,29$ потерь нет во всех моделях.

При сравнении типов обслуживания задач можно отметить, что беспriorитетный тип обслуживания обеспечивает наиболее равномерные значения среднего времени отклика по приоритетам, но практически не учитывает важность заявок. Абсолютный тип обслуживания сильнее выделяет высокоpriorитетные задачи, однако ухудшает обслуживание низкоpriorитетных задач. Относительный тип обслуживания является наиболее сбалансированным, так как учитывает приоритеты без резкого ухудшения характеристик задач с низким приоритетом.

В целом наиболее эффективной по среднему времени отклика и среднему времени выполнения задачи является организация с глобальной очередью. По типу обслуживания наиболее универсальным является относительный тип обслуживания задач, так как он обеспечивает баланс между учетом приоритетов, потерями задач и временными характеристиками.

4 Экономика

Разработка программы имитационного моделирования многопроцессорных приоритетных планировщиков задач, выполненная в рамках дипломного проекта, представляет собой инженерную разработку аппаратно-ориентированной модели вычислительной системы. В работе были реализованы и исследованы планировщики с глобальной и локальными очередями, а также режимы обслуживания без приоритетов, с относительными и абсолютными приоритетами.

При коммерческой реализации проекта аналогичного уровня потребовалось бы привлечение нескольких специалистов: разработчика VHDL-моделей, инженера по тестированию и моделированию цифровых систем, системного аналитика и руководителя проекта. Основные трудозатраты связаны с разработкой архитектуры планировщика, описанием блоков генерации заявок, управления очередями, выбора задач, назначения их процессорам, обработки вытеснения и сбора статистики.

Срок выполнения проекта составляет около пяти календарных месяцев, что при стандартном рабочем графике соответствует примерно 100–110 рабочим дням. С учетом средней заработной платы специалистов в области цифрового проектирования фонд оплаты труда команды при коммерческой разработке мог бы составить около 2,0–2,3 млн рублей. Дополнительно необходимо учитывать страховые взносы работодателя, которые увеличивают затраты примерно на 600–700 тыс. рублей.

Кроме расходов на персонал, в стоимость проекта входят затраты на программные средства моделирования, вычислительную технику, хранение проектных данных, оформление документации, тестирование и повторную проверку моделей. Эти расходы можно оценить примерно в 250–400 тыс. рублей.

Таким образом, ориентировочная стоимость разработки программы имитационного моделирования многопроцессорных приоритетных планировщиков задач составляет около 3,0–3,5 млн рублей. Полученная оценка

показывает, что выполненная работа по объему и сложности соответствует полноценному инженерному проекту.

Экономическая значимость разработки заключается в том, что созданная модель позволяет исследовать различные варианты планировщиков без изготовления физического аппаратного устройства. Это снижает затраты на предварительное проектирование, помогает выявить недостатки архитектуры на раннем этапе и выбрать наиболее эффективный вариант организации очередей и режимов обслуживания задач.

Разработанная модель может применяться как учебный и исследовательский инструмент, а также служить основой для дальнейшего расширения: увеличения числа процессоров, добавления новых алгоритмов планирования, изменения параметров очередей и введения динамических приоритетов.

5 Безопасность жизнедеятельности

В процессе выполнения дипломного проекта основная часть работ связана с использованием персонального компьютера, программных средств моделирования и разработки VHDL-описаний. Поэтому при организации рабочего места необходимо учитывать требования охраны труда, предъявляемые к помещениям, оборудованным электронно-вычислительной техникой.

Работа с персональным компьютером должна выполняться в помещении, соответствующем санитарно-гигиеническим, эргономическим требованиям. Помещение должно иметь достаточный уровень естественного или искусственного освещения. При отсутствии естественного освещения необходимо обосновать возможность безопасного выполнения работ за счет применения системы искусственного освещения. Оконные проемы рекомендуется оборудовать регулируемыми устройствами, например жалюзи, шторами или занавесями, позволяющими снизить вероятность появления бликов на экране монитора.

Персональные компьютеры, используемые для разработки и моделирования аппаратных планировщиков задач, не допускается размещать в местах, где возможно попадание влаги. Рабочие места должны быть подключены к электрической сети с исправным защитным заземлением. Это снижает вероятность поражения электрическим током и повышает безопасность эксплуатации вычислительной техники. Также необходимо следить за тем, чтобы системный блок, монитор и другое оборудование не располагались вплотную к стенам или мебели. Расстояние от вентиляционных отверстий до ближайших поверхностей должно составлять не менее 10 см, что обеспечивает нормальный теплообмен и предотвращает перегрев оборудования.

Для поддержания нормальных условий труда в помещении с компьютерной техникой следует регулярно выполнять влажную уборку и проветривание. Проветривание рекомендуется проводить не реже одного раза в час, так как длительная работа вычислительной техники может ухудшать микроклимат

помещения. Оптимальная температура воздуха при работе с ПК составляет примерно от 19 до 21 °С.

Рабочий стол должен обеспечивать удобное размещение монитора, клавиатуры, мыши, документов и другого оборудования, используемого при разработке дипломного проекта. Конструкция стола должна позволять пользователю сохранять удобную рабочую позу и свободно выполнять операции, связанные с написанием кода, анализом временных диаграмм и обработкой результатов моделирования. Допускается использование различных типов рабочих столов, если они соответствуют требованиям эргономики и обеспечивают безопасную работу.

Организация рабочих мест с персональными компьютерами должна исключать чрезмерную плотность размещения оборудования. Расстояние между рабочими столами с видеомониторами должно быть не менее 2 м в направлении экранов и не менее 1,2 м между боковыми поверхностями мониторов. Размещать компьютерную технику следует на расстоянии не менее 1 м от источников тепла. Также необходимо исключить прямое попадание солнечных лучей на экран монитора, поскольку это приводит к появлению бликов и повышает зрительное напряжение.

Не допускается установка рабочих мест рядом с силовыми кабелями, высоковольтными трансформаторами и другим оборудованием, которое может создавать электромагнитные помехи. При прокладке кабелей питания и интерфейсных соединений следует избегать их размещения непосредственно на полу. Если такая прокладка необходима, кабели должны быть закрыты защитными коробами или каналами, чтобы исключить риск повреждения проводов и травмирования пользователя.

Экран монитора должен располагаться на удобном для зрения расстоянии. Рекомендуемое расстояние от глаз пользователя до экрана составляет 600–700 мм, но не менее 500 мм. При этом необходимо учитывать размер отображаемых символов, разрешение экрана и характер выполняемой работы. При разработке

VHDL-моделей, анализе исходного кода и просмотре временных диаграмм особенно важно обеспечить четкость изображения и отсутствие бликов.

Искусственное освещение в помещении должно быть равномерным. Освещенность рабочей поверхности в зоне размещения персонального компьютера рекомендуется поддерживать на уровне 300–500 лк. Светильники должны быть расположены так, чтобы не создавать отражений на экране монитора. Освещенность поверхности экрана не должна превышать 300 лк.

При длительной работе за компьютером необходимо соблюдать режим труда и отдыха. Если продолжительность работы составляет 8 часов, регламентированные перерывы устанавливаются в зависимости от категории выполняемых работ. Для работ I категории предусматриваются перерывы по 15 минут через 2 часа после начала смены и через 2 часа после обеденного перерыва. Для работ II категории рекомендуется делать перерывы по 15 минут через 2 часа после начала смены и через 1,5–2 часа после обеденного перерыва либо по 10 минут после каждого часа работы. Для работ III категории устанавливаются перерывы по 20 минут через 1,5–2 часа после начала смены и через 1,5–2 часа после обеденного перерыва либо по 15 минут после каждого часа работы.

Если работа за компьютером продолжается 12 часов, то в первые 8 часов перерывы организуются так же, как при обычной 8-часовой смене. В течение последних 4 часов независимо от категории работы необходимо делать перерыв каждый час продолжительностью 15 минут.

Соблюдение перечисленных требований позволяет снизить зрительное и мышечное утомление, уменьшить влияние неблагоприятных факторов при работе с персональным компьютером и обеспечить безопасные условия при выполнении дипломного проекта, связанного с разработкой и имитационным моделированием многопроцессорных приоритетных планировщиков задач.

Заключение

В результате выполненной работы были изучены методы планирования задач в многопроцессорных системах. Рассмотрены принципы организации глобальных и локальных очередей, а также особенности бесприоритетного обслуживания, обслуживания с относительными приоритетами и обслуживания с абсолютными приоритетами.

На основе рассмотренных алгоритмов были разработаны шесть моделей многопроцессорных планировщиков задач. Для каждого способа организации очередей были реализованы три режима обслуживания заявок: бесприоритетный режим, режим с относительными приоритетами и режим с абсолютными приоритетами. Таким образом, были получены модели планировщиков с глобальной очередью и локальными очередями для всех трёх вариантов обслуживания.

Для каждой модели было выполнено имитационное моделирование в среде Xilinx 9.1. В качестве оцениваемых характеристик использовались: общее число созданных заявок, число потерянных заявок, число потерянных заявок по каждому приоритету, среднее время пребывания заявок в системе, среднее время пребывания заявок каждого приоритета, средняя длина очереди и загрузка процессоров.

Также были рассмотрены режимы с двумя приоритетами заявок 0 и 1 при разных соотношениях входного потока и ограничениях мест в очереди. Это позволило оценить, при каких условиях система работает без потерь заявок.

Сравнение разработанных моделей показало, что организация очередей существенно влияет на характеристики работы СМО. Глобальная очередь обеспечивает более равномерное использование процессоров, так как все процессоры обращаются к единой очереди заявок. Локальные очереди позволяют распределить заявки между отдельными процессорами, но при этом возможно неравномерное заполнение очередей, что влияет на число отказов и среднее время пребывания заявок в системе.

Режим обслуживания также влияет на распределение заявок между приоритетами. При абсолютных приоритетах более важные заявки получают преимущество за счёт возможности вытеснения менее приоритетных задач. При относительных приоритетах вытеснение не выполняется, поэтому уже начатая задача завершается без прерывания. В бесприоритетном режиме заявки обслуживаются без учёта их приоритета.

Полученные результаты могут быть использованы при разработке и сравнении аппаратно ориентированных многопроцессорных планировщиков задач. Дальнейшее развитие работы может быть связано с расширением числа процессоров, увеличением числа приоритетов, изменением правил распределения заявок между очередями и переносом моделей на современные ПЛИС и системы на кристалле.

Список использованных источников

1. Вашкевич, Н. П. Недетерминированные автоматы и их использование для реализации систем параллельной обработки информации / Н. П. Вашкевич, Р. А. Бикташев. – Пенза: Изд-во ПГУ, 2016. – 394 с. [Электронный ресурс] // URL: <https://elib.pnzgu.ru/files/eb/nXj55aMRlAdj.pdf> (дата обращения: 06.03.2026).
2. Таненбаум, Э. Современные операционные системы / Э. Таненбаум, Х. Бос. – 4-е изд. – Санкт-Петербург: Питер, 2015. – 1120 с. [Электронный ресурс] // URL: <https://library.tsilikin.ru/Техника/Информационные технологии / OS / Современные операционные системы.pdf> (дата обращения: 13.02.2026).
3. Столлинс, У. Операционные системы. Внутреннее устройство и принципы проектирования / У. Столлинс. – 9-е изд. – М.: Вильямс, 2018. – 960 с.
4. Ашенден, П. Дж. Проектирование цифровых устройств на VHDL. Руководство разработчика / П. Дж. Ашенден. – 3-е изд. – М.: ДМК Пресс, 2011. – 772 с.
5. Таненбаум, Э. Многоуровневая организация ЭВМ / Э. Таненбаум. – М.: Мир, 1979. – 547 с. [Электронный ресурс] // URL: <https://publ.lib.ru/ARCHIVES/T/TANENBAUMEndryuStyuart/TanenbaumE.S.html> (дата обращения: 24.02.2026).
6. Вашкевич, Н. П. Недетерминированные автоматы в проектировании систем параллельной обработки: учеб. пособие / Н. П. Вашкевич. – Пенза: Изд-во Пенз. гос. ун-та, 2004. – 280 с. [Электронный ресурс] // URL: <https://is.ifmo.ru/automata/vashkevich-nda.pdf> (дата обращения: 18.03.2026).
7. Вашкевич, Н. П. Аппаратная реализация функций синхронизации параллельных процессов при обращении к разделяемому ресурсу на основе ПЛИС / Н. П. Вашкевич, Р. А. Бикташев, Е. И. Гурин // Известия вузов. Поволжский регион. Технические науки. – 2007. – № 2. – С. 3–12. [Электронный ресурс] // URL: <https://cyberleninka.ru/article/n/apparatnaya-realizatsiya-funktsiy-sinhronizatsii-parallelnyh-protseessov-pri-obraschenii-k-razdelyaemomu-resursu-na-osnove-plis> (дата обращения: 27.03.2026).

8. Алиев, Т. И. Основы моделирования дискретных систем / Т. И. Алиев. – Санкт-Петербург: СПбГУ ИТМО, 2009. – 363 с. [Электронный ресурс] // URL: https://books.ifmo.ru/book/445/osnov_modelirovan_diskretny_sistem.html (дата обращения: 08.04.2026).

9. Бикташев, Р. А. Сравнительный анализ производительности средств управления задачами методом моделирования / Р. А. Бикташев, Б. Д. Королев // Сборник научных статей Всероссийской межвузовской научно-практической конференции / под ред. Л. Р. Фионовой. – 2019.

10. Кларк, Э. М. Верификация моделей программы: Model Checking / Э. М. Кларк, О. Грамберг, Д. Пелед; пер. с англ. – М.: МЦНМО, 2002. – 416 с. [Электронный ресурс] // URL: <https://biblio.mccme.ru/node/1690> (дата обращения: 25.04.2026).

Приложение А

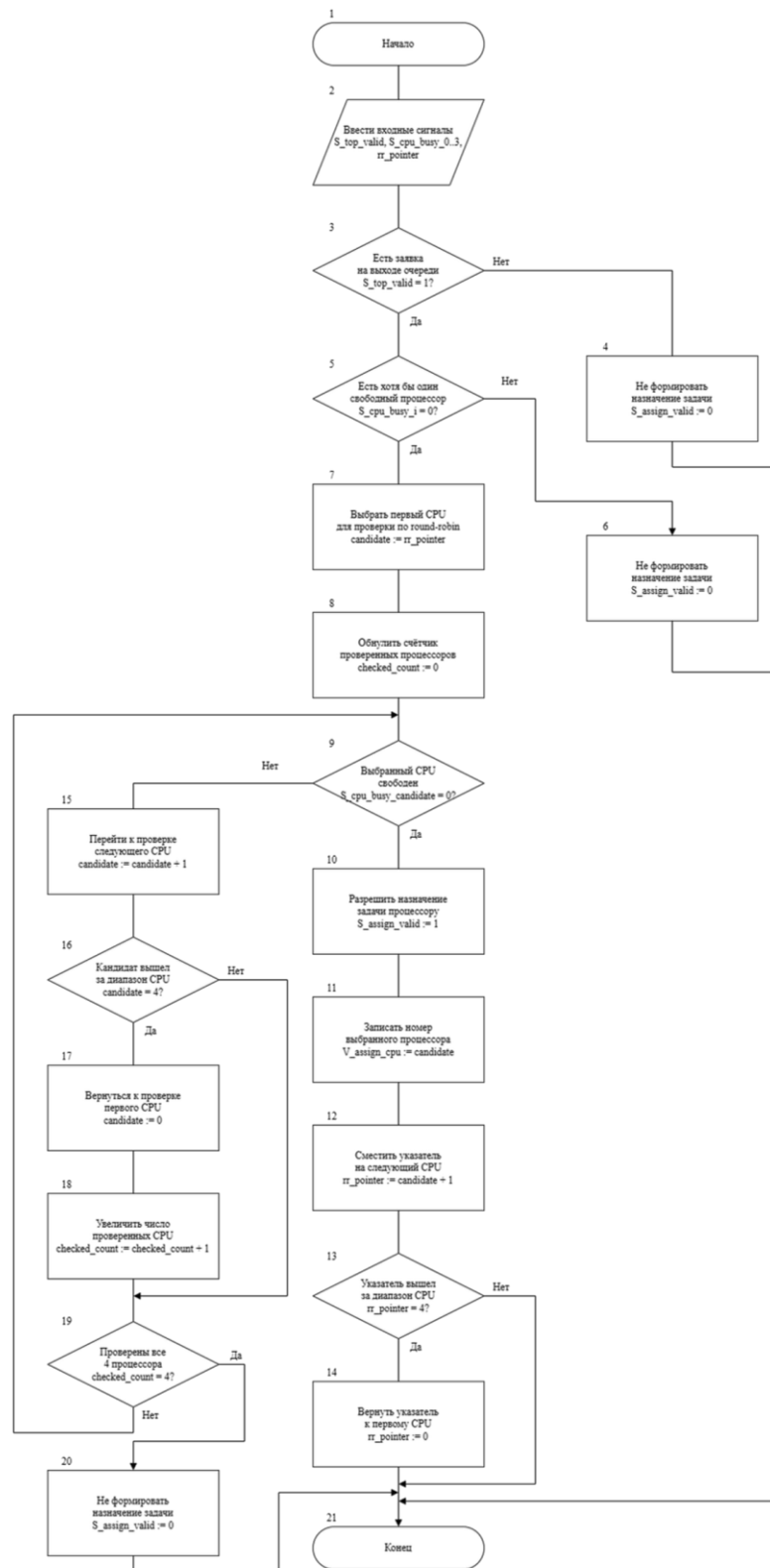
(обязательное)

Программа имитационного моделирования многопроцессорных приоритетных
планировщиков задач

Схемы программы

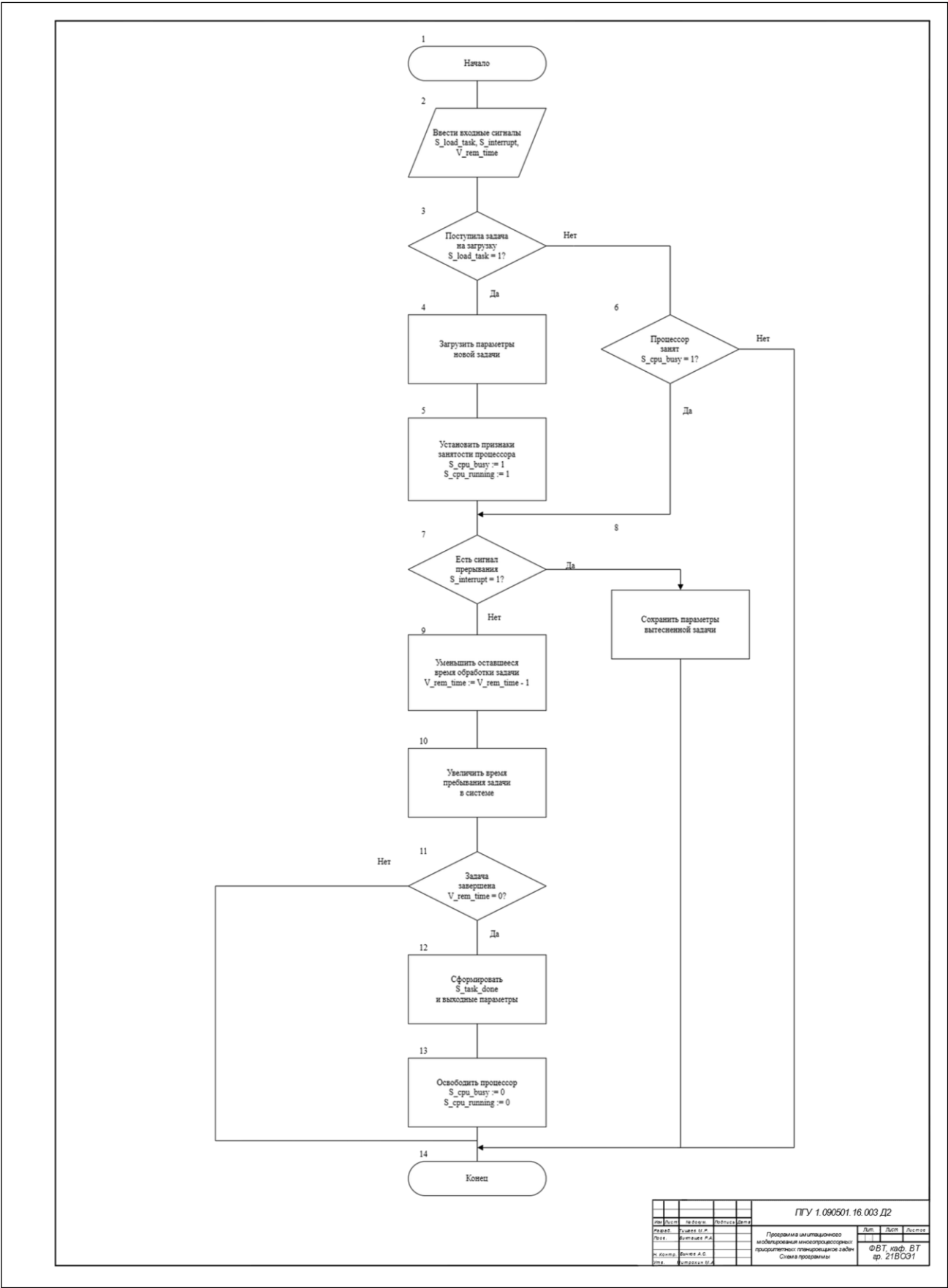
					ПГУ 1.090501.05.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		113

Схема программы выбора процессора для глобальной очереди



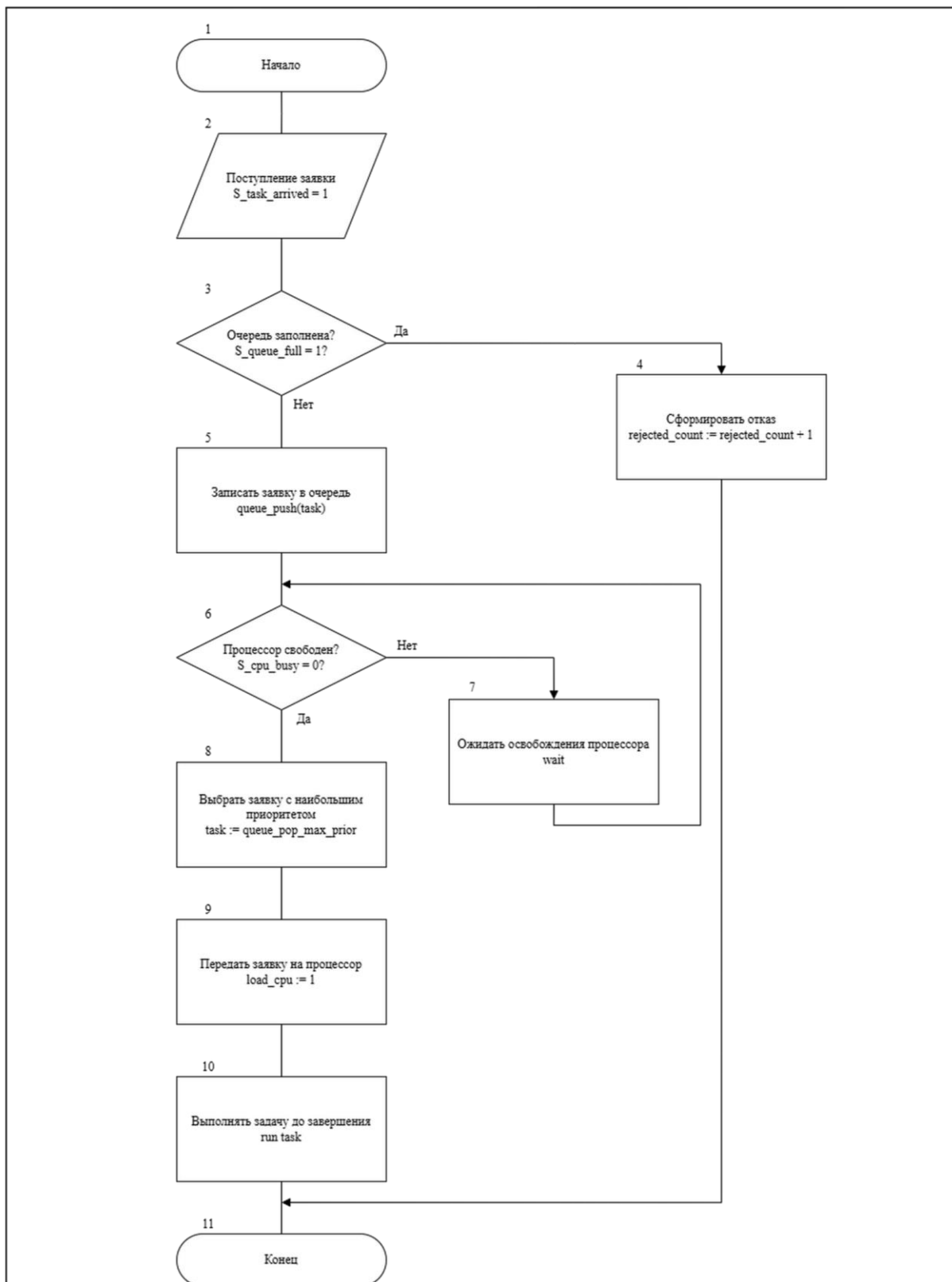
					ПГУ 1.090501.16.002 Д2		
Имя	Фамилия	Инициалы	Подпись	Дата			
Решет Павел	Тютин Игорь И.П.	Васильев И.А.	Программа имитационного моделирования многопроцессорных приоритетных планировщиков задач Схема программы		Лист	Лист	Листов
И. контр.	Васильев А.С.					ФВТ, каф. ВТ ар. 21В031	

Схема программы выполнения задачи процессорным блоком



ПГУ 1.090501.16.003 Д2									
Исполн.	Исполн.	Исполн.	Исполн.	Исполн.	Исполн.	Исполн.	Исполн.	Исполн.	Исполн.
Разраб.	Разраб.	Разраб.	Разраб.	Разраб.	Разраб.	Разраб.	Разраб.	Разраб.	Разраб.
Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.
И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.	И. Контр.
Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.	Програ.

Схема программы обслуживания задач для относительных приоритетов



ПГУ 1.090501.16.004 Д2									
Изм.	Лист	№ докум.	Подпись	Дата	Программа имитационного моделирования многопроцессорной приоритетной планирующей задачи				
Изм.	Лист	№ докум.	Подпись	Дата	ФВТ, каф. ВТ пр. 21В031				

Схема программы обслуживания задач для абсолютных приоритетов

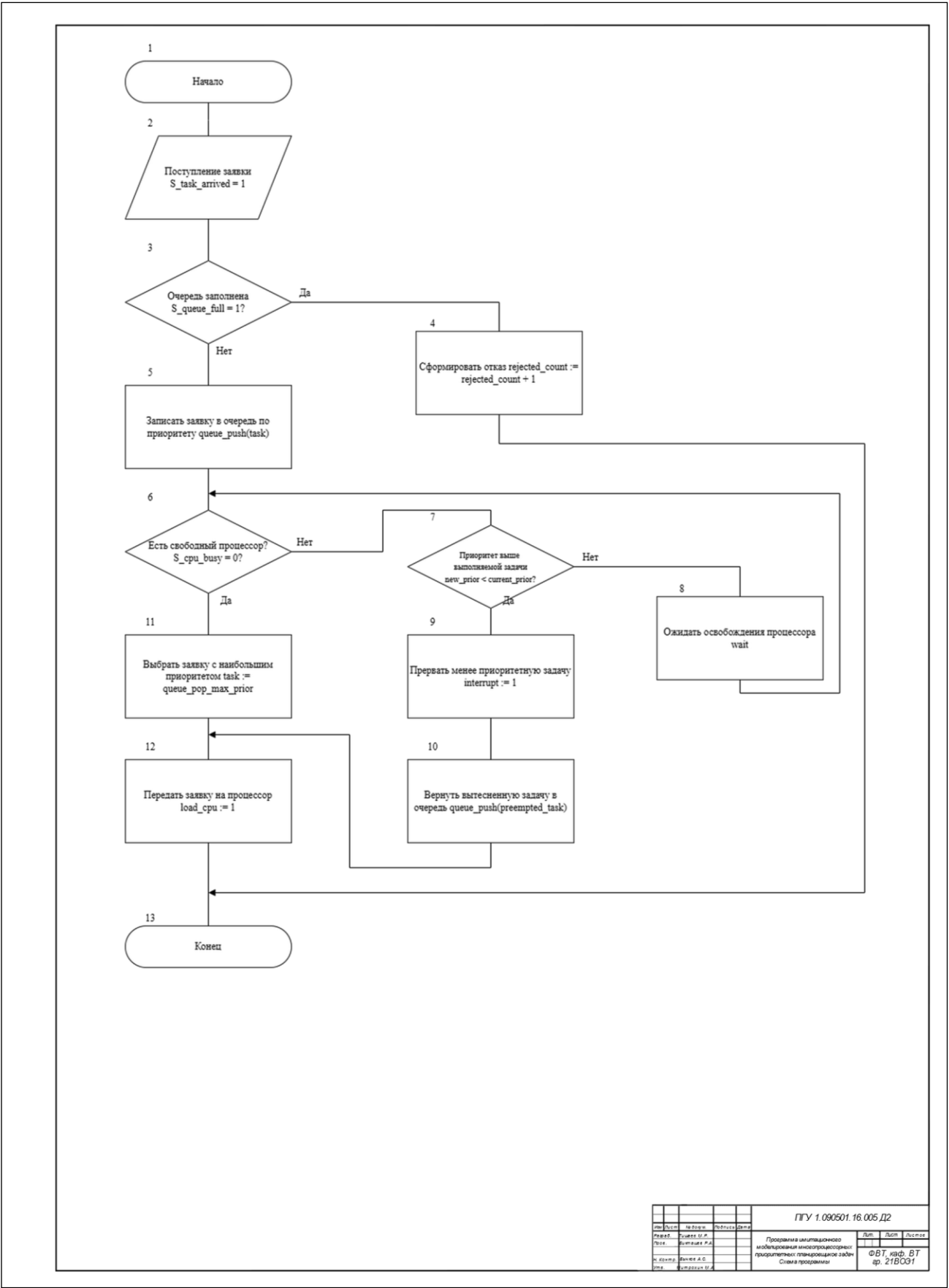
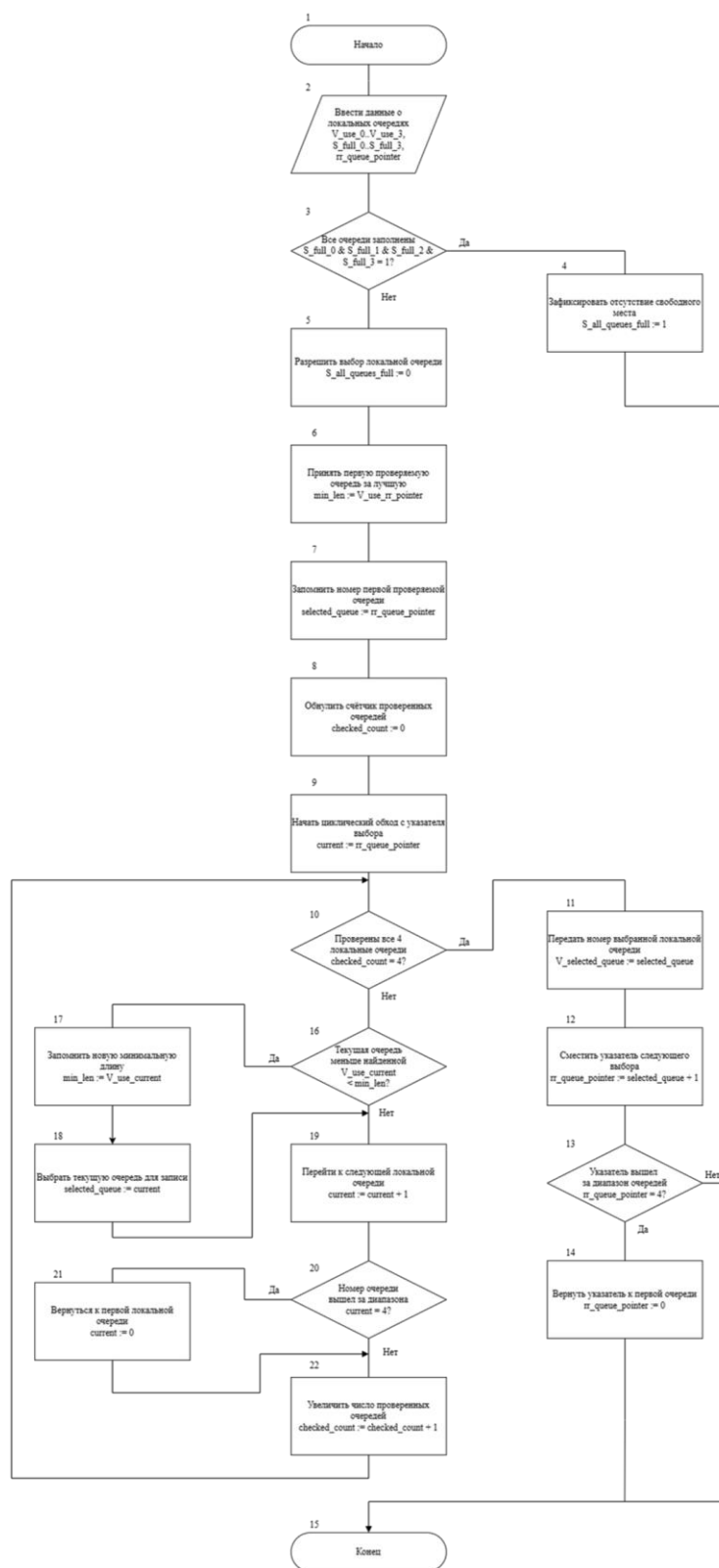


Схема программы выбора локальной очереди



					ПГУ 1.090501.16.006 Д2								
Имя	Фамилия	Название	Подпись	Дата						Лит.	Дост.	Листов	
Иванов	Сергей И.А.	Выпуск 18			Программа имитационного моделирования микропроцессоров приоритетных планирующих задач Схема программы						ФВТ, каф. ВТ стр. 21B031		
И. Кондр.	Васильев А.С.	Выпуск 18											

Приложение Б

(обязательное)

Программа имитационного моделирования многопроцессорных приоритетных
планировщиков задач

Листинг программы

					ПГУ 1.090501.05.001 ПЗ	Лист
						119
Изм.	Лист	№ докум.	Подпись	Дата		

```

block_gen_zadach.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
entity block_gen_zadach is
    generic (
        LAMBDA_C      : real := 0.4;
        ARRIVAL_SEED1_C : positive := 101;
        ARRIVAL_SEED2_C : positive := 203
    );
    port (
        CLK          : in  std_logic;
        reset        : in  std_logic;
        start        : in  std_logic;
        V_task_id     : out std_logic_vector(15 downto 0);
        V_task_prior  : out std_logic_vector(3  downto 0);
        S_task_arrived : out std_logic
    );
end entity block_gen_zadach;
architecture behavioral of block_gen_zadach is
    signal task_id_reg      : unsigned(15 downto 0) := (others => '0');
    signal task_prior_reg   : std_logic_vector(3  downto 0) := (others => '0');
    signal arrived_reg      : std_logic := '0';
    signal interval_timer   : integer := 0;
begin
    V_task_id      <= std_logic_vector(task_id_reg);
    V_task_prior   <= task_prior_reg;
    S_task_arrived <= arrived_reg;
    process (CLK)
        variable seed1      : positive := ARRIVAL_SEED1_C;
        variable seed2      : positive := ARRIVAL_SEED2_C;
        variable priority_real : real;
        variable priority_int  : integer;
        variable rand        : real;
        variable interval_real : real;
        variable interval_ticks : integer;
    begin
        if rising_edge(CLK) then
            arrived_reg <= '0';
            if reset = '1' then
                seed1 := ARRIVAL_SEED1_C;
                seed2 := ARRIVAL_SEED2_C;
                task_id_reg <= (others => '0');
                task_prior_reg <= (others => '0');
                arrived_reg <= '0';
                interval_timer <= 0;
            elsif start = '1' then
                if interval_timer <= 0 then
                    uniform(seed1, seed2, priority_real);
                    priority_int := integer(floor(priority_real * 16.0));
                    if priority_int < 0 then
                        priority_int := 0;
                    elsif priority_int > 15 then
                        priority_int := 15;
                    end if;
                    task_id_reg <= task_id_reg + 1;
                    task_prior_reg <= std_logic_vector(to_unsigned(priority_int,
4));
                    arrived_reg <= '1';
                    uniform(seed1, seed2, rand);
                    interval_real := -1.0 / LAMBDA_C * log(1.0 - rand);
                    interval_ticks := integer(floor(interval_real));

```

					ПГУ 1.090501.05.001 ПЗ	Лист
						120
Изм.	Лист	№ докум.	Подпись	Дата		

```

        if interval_ticks <= 0 then
            interval_ticks := 1;
        end if;
        interval_timer <= interval_ticks - 1;
    else
        interval_timer <= interval_timer - 1;
    end if;
else
    arrived_reg <= '0';
end if;
end if;
end process;
end architecture behavioral;
block_upr_ocher_global_abs.vhd
library ieee;
use ieee.std_logic_1164.all;
entity block_upr_ocher_global_abs is
    generic (
        QUEUE_SIZE_C : integer := 16
    );
    port (
        CLK           : in  std_logic;
        reset         : in  std_logic;
        S_task_arrived : in  std_logic;
        V_task_id_gen  : in  std_logic_vector(15 downto 0);
        V_task_prior_gen : in  std_logic_vector(3 downto 0);
        S_queue_full   : in  std_logic;
        S_all_cpu_busy  : in  std_logic;
        S_preempted_valid_0 : in  std_logic;
        S_preempted_valid_1 : in  std_logic;
        S_preempted_valid_2 : in  std_logic;
        S_preempted_valid_3 : in  std_logic;
        S_pop_top_0      : in  std_logic;
        S_pop_top_1      : in  std_logic;
        S_pop_top_2      : in  std_logic;
        S_pop_top_3      : in  std_logic;
        V_queue_length   : in  integer;
        S_new_wrreq      : out std_logic;
        V_new_task_id     : out std_logic_vector(15 downto 0);
        V_new_task_prior  : out std_logic_vector(3 downto 0);
        S_refuse         : out std_logic;
        V_refused_prior   : out std_logic_vector(3 downto 0);
        S_task_created    : out std_logic;
        V_created_prior   : out std_logic_vector(3 downto 0)
    );
end entity block_upr_ocher_global_abs;
architecture behavioral of block_upr_ocher_global_abs is
    signal new_wrreq_reg      : std_logic := '0';
    signal new_task_id_reg    : std_logic_vector(15 downto 0) := (others => '0');
    signal new_task_prior_reg : std_logic_vector(3 downto 0) := (others => '0');
    signal refuse_reg        : std_logic := '0';
    signal refused_prior_reg  : std_logic_vector(3 downto 0) := (others => '0');
    signal created_reg       : std_logic := '0';
    signal created_prior_reg  : std_logic_vector(3 downto 0) := (others => '0');
begin
    S_new_wrreq      <= new_wrreq_reg;
    V_new_task_id    <= new_task_id_reg;
    V_new_task_prior <= new_task_prior_reg;
    S_refuse         <= refuse_reg;
    V_refused_prior  <= refused_prior_reg;
    S_task_created   <= created_reg;
    V_created_prior  <= created_prior_reg;
    process (CLK)

```

```

variable effective_free_slots : integer;
variable pop_count           : integer;
variable preempted_count     : integer;
begin
  if rising_edge(CLK) then
    new_wrreq_reg <= '0';
    refuse_reg    <= '0';
    created_reg   <= '0';
    if reset = '1' then
      new_task_id_reg    <= (others => '0');
      new_task_prior_reg <= (others => '0');
      refused_prior_reg  <= (others => '0');
      created_prior_reg  <= (others => '0');
    elsif S_task_arrived = '1' then
      created_reg        <= '1';
      created_prior_reg  <= V_task_prior_gen;
      pop_count := 0;
      if S_pop_top_0 = '1' and V_queue_length > 0 then
        pop_count := 1;
        if S_pop_top_1 = '1' and V_queue_length > 1 then
          pop_count := 2;
          if S_pop_top_2 = '1' and V_queue_length > 2 then
            pop_count := 3;
            if S_pop_top_3 = '1' and V_queue_length > 3 then
              pop_count := 4;
            end if;
          end if;
        end if;
      end if;
      preempted_count := 0;
      if S_preempted_valid_0 = '1' then
        preempted_count := preempted_count + 1;
      end if;
      if S_preempted_valid_1 = '1' then
        preempted_count := preempted_count + 1;
      end if;
      if S_preempted_valid_2 = '1' then
        preempted_count := preempted_count + 1;
      end if;
      if S_preempted_valid_3 = '1' then
        preempted_count := preempted_count + 1;
      end if;
      effective_free_slots := QUEUE_SIZE_C - V_queue_length + pop_count -
preempted_count;
      if effective_free_slots > 0 then
        new_wrreq_reg    <= '1';
        new_task_id_reg   <= V_task_id_gen;
        new_task_prior_reg <= V_task_prior_gen;
      else
        refuse_reg        <= '1';
        refused_prior_reg <= V_task_prior_gen;
      end if;
    end if;
  end if;
end process;
end architecture behavioral;
block_oher_global_abs.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity block_oher_global_abs is
  generic (
    QUEUE_SIZE_C : integer := 16

```

```

);
port (
    CLK                      : in  std_logic;
    reset                    : in  std_logic;
    S_wrreq_preempted_0     : in  std_logic;
    V_preempted_task_id_0   : in  std_logic_vector(15 downto 0);
    V_preempted_prior_0    : in  std_logic_vector(3 downto 0);
    V_preempted_wait_time_0 : in  integer;
    V_preempted_system_time_0 : in integer;
    V_preempted_rem_time_0  : in  integer;
    V_preempted_started_0  : in  std_logic;
    S_wrreq_preempted_1     : in  std_logic;
    V_preempted_task_id_1   : in  std_logic_vector(15 downto 0);
    V_preempted_prior_1    : in  std_logic_vector(3 downto 0);
    V_preempted_wait_time_1 : in  integer;
    V_preempted_system_time_1 : in integer;
    V_preempted_rem_time_1  : in  integer;
    V_preempted_started_1  : in  std_logic;
    S_wrreq_preempted_2     : in  std_logic;
    V_preempted_task_id_2   : in  std_logic_vector(15 downto 0);
    V_preempted_prior_2    : in  std_logic_vector(3 downto 0);
    V_preempted_wait_time_2 : in  integer;
    V_preempted_system_time_2 : in integer;
    V_preempted_rem_time_2  : in  integer;
    V_preempted_started_2  : in  std_logic;
    S_wrreq_preempted_3     : in  std_logic;
    V_preempted_task_id_3   : in  std_logic_vector(15 downto 0);
    V_preempted_prior_3    : in  std_logic_vector(3 downto 0);
    V_preempted_wait_time_3 : in  integer;
    V_preempted_system_time_3 : in integer;
    V_preempted_rem_time_3  : in  integer;
    V_preempted_started_3  : in  std_logic;
    S_wrreq_new             : in  std_logic;
    V_new_task_id           : in  std_logic_vector(15 downto 0);
    V_new_prior             : in  std_logic_vector(3 downto 0);
    S_pop_top_0             : in  std_logic;
    S_pop_top_1             : in  std_logic;
    S_pop_top_2             : in  std_logic;
    S_pop_top_3             : in  std_logic;
    S_queue_full            : out std_logic;
    S_queue_empty           : out std_logic;
    V_queue_length          : out integer;
    S_top_valid_0           : out std_logic;
    V_top_task_id_0         : out std_logic_vector(15 downto 0);
    V_top_prior_0           : out std_logic_vector(3 downto 0);
    V_top_wait_time_0       : out integer;
    V_top_system_time_0     : out integer;
    V_top_rem_time_0        : out integer;
    V_top_started_0        : out std_logic;
    S_top_valid_1           : out std_logic;
    V_top_task_id_1         : out std_logic_vector(15 downto 0);
    V_top_prior_1           : out std_logic_vector(3 downto 0);
    V_top_wait_time_1       : out integer;
    V_top_system_time_1     : out integer;
    V_top_rem_time_1        : out integer;
    V_top_started_1        : out std_logic;
    S_top_valid_2           : out std_logic;
    V_top_task_id_2         : out std_logic_vector(15 downto 0);
    V_top_prior_2           : out std_logic_vector(3 downto 0);
    V_top_wait_time_2       : out integer;
    V_top_system_time_2     : out integer;
    V_top_rem_time_2        : out integer;
    V_top_started_2        : out std_logic;

```

```

        S_top_valid_3          : out std_logic;
        V_top_task_id_3        : out std_logic_vector(15 downto 0);
        V_top_prior_3          : out std_logic_vector(3 downto 0);
        V_top_wait_time_3      : out integer;
        V_top_system_time_3    : out integer;
        V_top_rem_time_3       : out integer;
        V_top_started_3        : out std_logic;
        S_task_removed          : out std_logic
    );
end entity block_ocher_global_abs;
architecture behavioral of block_ocher_global_abs is
    subtype queue_index_t is integer range 0 to 15;
    type slv16_array_t is array (queue_index_t) of std_logic_vector(15 downto 0);
    type slv4_array_t is array (queue_index_t) of std_logic_vector(3 downto 0);
    type int_array_t is array (queue_index_t) of integer;
    type sl_array_t is array (queue_index_t) of std_logic;
    signal q_task_id : slv16_array_t := (others => (others => '0'));
    signal q_prior : slv4_array_t := (others => (others => '0'));
    signal q_wait : int_array_t := (others => 0);
    signal q_system : int_array_t := (others => 0);
    signal q_rem : int_array_t := (others => 0);
    signal q_started : sl_array_t := (others => '0');
    signal q_seq : int_array_t := (others => 0);
    signal q_count : integer range 0 to 16 := 0;
    signal seq_next : integer := 0;
    signal removed_reg : std_logic := '0';
    procedure insert_task_sorted(
        variable n_task_id : inout slv16_array_t;
        variable n_prior : inout slv4_array_t;
        variable n_wait : inout int_array_t;
        variable n_system : inout int_array_t;
        variable n_rem : inout int_array_t;
        variable n_started : inout sl_array_t;
        variable n_seq : inout int_array_t;
        variable n_count : inout integer;
        variable n_seq_next : inout integer;
        constant task_id_in : in std_logic_vector(15 downto 0);
        constant prior_in : in std_logic_vector(3 downto 0);
        constant wait_in : in integer;
        constant system_in : in integer;
        constant rem_in : in integer;
        constant started_in : in std_logic
    ) is
        variable insert_pos : integer;
    begin
        if n_count >= QUEUE_SIZE_C then
            return;
        end if;
        insert_pos := n_count;
        for i in 0 to 15 loop
            if i < n_count then
                if unsigned(n_prior(i)) > unsigned(prior_in) then
                    insert_pos := i;
                    exit;
                end if;
            end if;
        end loop;
        for i in 15 downto 1 loop
            if i > insert_pos and i <= n_count then
                n_task_id(i) := n_task_id(i - 1);
                n_prior(i) := n_prior(i - 1);
                n_wait(i) := n_wait(i - 1);
                n_system(i) := n_system(i - 1);
            end if;
        end loop;
    end insert_task_sorted;

```

```

        n_rem(i)      := n_rem(i - 1);
        n_started(i) := n_started(i - 1);
        n_seq(i)      := n_seq(i - 1);
    end if;
end loop;
n_task_id(insert_pos) := task_id_in;
n_prior(insert_pos)  := prior_in;
n_wait(insert_pos)   := wait_in;
n_system(insert_pos) := system_in;
n_rem(insert_pos)    := rem_in;
n_started(insert_pos) := started_in;
n_seq(insert_pos)    := n_seq_next;
n_seq_next := n_seq_next + 1;
n_count    := n_count + 1;
end procedure;
begin
    S_queue_full  <= '1' when q_count >= QUEUE_SIZE_C else '0';
    S_queue_empty <= '1' when q_count = 0 else '0';
    V_queue_length <= q_count;
    S_task_removed <= removed_reg;
    S_top_valid_0 <= '1' when q_count > 0 else '0';
    S_top_valid_1 <= '1' when q_count > 1 else '0';
    S_top_valid_2 <= '1' when q_count > 2 else '0';
    S_top_valid_3 <= '1' when q_count > 3 else '0';
    V_top_task_id_0 <= q_task_id(0);
    V_top_prior_0 <= q_prior(0);
    V_top_wait_time_0 <= q_wait(0);
    V_top_system_time_0 <= q_system(0);
    V_top_rem_time_0 <= q_rem(0);
    V_top_started_0 <= q_started(0);
    V_top_task_id_1 <= q_task_id(1);
    V_top_prior_1 <= q_prior(1);
    V_top_wait_time_1 <= q_wait(1);
    V_top_system_time_1 <= q_system(1);
    V_top_rem_time_1 <= q_rem(1);
    V_top_started_1 <= q_started(1);
    V_top_task_id_2 <= q_task_id(2);
    V_top_prior_2 <= q_prior(2);
    V_top_wait_time_2 <= q_wait(2);
    V_top_system_time_2 <= q_system(2);
    V_top_rem_time_2 <= q_rem(2);
    V_top_started_2 <= q_started(2);
    V_top_task_id_3 <= q_task_id(3);
    V_top_prior_3 <= q_prior(3);
    V_top_wait_time_3 <= q_wait(3);
    V_top_system_time_3 <= q_system(3);
    V_top_rem_time_3 <= q_rem(3);
    V_top_started_3 <= q_started(3);
    process (CLK)
        variable n_task_id : slv16_array_t;
        variable n_prior   : slv4_array_t;
        variable n_wait    : int_array_t;
        variable n_system  : int_array_t;
        variable n_rem     : int_array_t;
        variable n_started : sl_array_t;
        variable n_seq     : int_array_t;
        variable n_count   : integer;
        variable n_seq_next : integer;
        variable pop_count : integer;
    begin
        if rising_edge(CLK) then
            removed_reg <= '0';
            if reset = '1' then

```

```

q_task_id <= (others => (others => '0'));
q_prior   <= (others => (others => '0'));
q_wait    <= (others => 0);
q_system  <= (others => 0);
q_rem     <= (others => 0);
q_started <= (others => '0');
q_seq     <= (others => 0);
q_count   <= 0;
seq_next  <= 0;
else
  n_task_id := q_task_id;
  n_prior   := q_prior;
  n_wait    := q_wait;
  n_system  := q_system;
  n_rem     := q_rem;
  n_started := q_started;
  n_seq     := q_seq;
  n_count   := q_count;
  n_seq_next := seq_next;
  for i in 0 to 15 loop
    if i < n_count then
      n_wait(i) := n_wait(i) + 1;
      n_system(i) := n_system(i) + 1;
    end if;
  end loop;
  pop_count := 0;
  if S_pop_top_0 = '1' and n_count > 0 then
    pop_count := 1;
    if S_pop_top_1 = '1' and n_count > 1 then
      pop_count := 2;
      if S_pop_top_2 = '1' and n_count > 2 then
        pop_count := 3;
        if S_pop_top_3 = '1' and n_count > 3 then
          pop_count := 4;
        end if;
      end if;
    end if;
  end if;
  if pop_count > 0 then
    for i in 0 to 15 loop
      if i + pop_count < n_count then
        n_task_id(i) := n_task_id(i + pop_count);
        n_prior(i)   := n_prior(i + pop_count);
        n_wait(i)    := n_wait(i + pop_count);
        n_system(i)  := n_system(i + pop_count);
        n_rem(i)     := n_rem(i + pop_count);
        n_started(i) := n_started(i + pop_count);
        n_seq(i)     := n_seq(i + pop_count);
      end if;
    end loop;
    n_count := n_count - pop_count;
    removed_reg <= '1';
  end if;
  if S_wrrreq_preempted_0 = '1' and n_count < QUEUE_SIZE_C then
    insert_task_sorted(n_task_id, n_prior, n_wait, n_system, n_rem,
n_started, n_seq, n_count, n_seq_next,
V_preempted_task_id_0, V_preempted_prior_0,
V_preempted_wait_time_0,
V_preempted_system_time_0, V_preempted_rem_time_0,
V_preempted_started_0);
  end if;
  if S_wrrreq_preempted_1 = '1' and n_count < QUEUE_SIZE_C then

```

```

        insert_task_sorted(n_task_id, n_prior, n_wait, n_system, n_rem,
n_started, n_seq, n_count, n_seq_next,
                        V_preempted_task_id_1,                        V_preempted_prior_1,
V_preempted_wait_time_1,                        V_preempted_system_time_1,                        V_preempted_rem_time_1,
V_preempted_started_1);
        end if;
        if S_wrreq_preempted_2 = '1' and n_count < QUEUE_SIZE_C then
            insert_task_sorted(n_task_id, n_prior, n_wait, n_system, n_rem,
n_started, n_seq, n_count, n_seq_next,
                        V_preempted_task_id_2,                        V_preempted_prior_2,
V_preempted_wait_time_2,                        V_preempted_system_time_2,                        V_preempted_rem_time_2,
V_preempted_started_2);
        end if;
        if S_wrreq_preempted_3 = '1' and n_count < QUEUE_SIZE_C then
            insert_task_sorted(n_task_id, n_prior, n_wait, n_system, n_rem,
n_started, n_seq, n_count, n_seq_next,
                        V_preempted_task_id_3,                        V_preempted_prior_3,
V_preempted_wait_time_3,                        V_preempted_system_time_3,                        V_preempted_rem_time_3,
V_preempted_started_3);
        end if;
        if S_wrreq_new = '1' and n_count < QUEUE_SIZE_C then
            insert_task_sorted(n_task_id, n_prior, n_wait, n_system, n_rem,
n_started, n_seq, n_count, n_seq_next,
                        V_new_task_id, V_new_prior, 0, 0, 0, '0');
        end if;
        for i in 0 to 15 loop
            if i >= n_count then
                n_task_id(i) := (others => '0');
                n_prior(i)   := (others => '0');
                n_wait(i)    := 0;
                n_system(i)  := 0;
                n_rem(i)     := 0;
                n_started(i) := '0';
                n_seq(i)     := 0;
            end if;
        end loop;
        q_task_id <= n_task_id;
        q_prior   <= n_prior;
        q_wait    <= n_wait;
        q_system  <= n_system;
        q_rem     <= n_rem;
        q_started <= n_started;
        q_seq     <= n_seq;
        q_count   <= n_count;
        seq_next  <= n_seq_next;
    end if;
end if;
end process;
end architecture behavioral;
block_vibora_proc_global_abs.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity block_vibora_proc_global_abs is
    port (
        CLK                : in  std_logic;
        reset               : in  std_logic;
        S_cpu_busy_0        : in  std_logic;
        S_cpu_busy_1        : in  std_logic;
        S_cpu_busy_2        : in  std_logic;

```

```

S_cpu_busy_3           : in  std_logic;
S_cpu_running_0        : in  std_logic;
S_cpu_running_1        : in  std_logic;
S_cpu_running_2        : in  std_logic;
S_cpu_running_3        : in  std_logic;
V_cpu_prior_0          : in  std_logic_vector(3 downto 0);
V_cpu_prior_1          : in  std_logic_vector(3 downto 0);
V_cpu_prior_2          : in  std_logic_vector(3 downto 0);
V_cpu_prior_3          : in  std_logic_vector(3 downto 0);
S_top_valid_0          : in  std_logic;
V_top_prior_0          : in  std_logic_vector(3 downto 0);
S_top_valid_1          : in  std_logic;
V_top_prior_1          : in  std_logic_vector(3 downto 0);
S_top_valid_2          : in  std_logic;
V_top_prior_2          : in  std_logic_vector(3 downto 0);
S_top_valid_3          : in  std_logic;
V_top_prior_3          : in  std_logic_vector(3 downto 0);
S_assign_valid_0        : out std_logic;
V_assign_cpu_0          : out integer range 0 to 3;
S_assign_interrupt_0    : out std_logic;
S_assign_valid_1        : out std_logic;
V_assign_cpu_1          : out integer range 0 to 3;
S_assign_interrupt_1    : out std_logic;
S_assign_valid_2        : out std_logic;
V_assign_cpu_2          : out integer range 0 to 3;
S_assign_interrupt_2    : out std_logic;
S_assign_valid_3        : out std_logic;
V_assign_cpu_3          : out integer range 0 to 3;
S_assign_interrupt_3    : out std_logic
);
end entity block_vibora_proc_global_abs;
architecture behavioral of block_vibora_proc_global_abs is
    subtype prior_vec_t is std_logic_vector(3 downto 0);
    type sl_array_t is array (0 to 3) of std_logic;
    type int_array_t is array (0 to 3) of integer;
    signal rr_pointer : integer range 0 to 3 := 3;
    signal assign_valid_0_reg : std_logic := '0';
    signal assign_valid_1_reg : std_logic := '0';
    signal assign_valid_2_reg : std_logic := '0';
    signal assign_valid_3_reg : std_logic := '0';
    signal assign_cpu_0_reg : integer range 0 to 3 := 0;
    signal assign_cpu_1_reg : integer range 0 to 3 := 0;
    signal assign_cpu_2_reg : integer range 0 to 3 := 0;
    signal assign_cpu_3_reg : integer range 0 to 3 := 0;
    signal assign_interrupt_0_reg : std_logic := '0';
    signal assign_interrupt_1_reg : std_logic := '0';
    signal assign_interrupt_2_reg : std_logic := '0';
    signal assign_interrupt_3_reg : std_logic := '0';
    signal last_cpu_next : integer range 0 to 3 := 3;
    function busy_at(
        idx : integer;
        b0 : std_logic; b1 : std_logic; b2 : std_logic; b3 : std_logic
    ) return std_logic is
    begin
        case idx is
            when 0 => return b0;
            when 1 => return b1;
            when 2 => return b2;
            when others => return b3;
        end case;
    end function;
    function running_at(
        idx : integer;

```

```

    r0 : std_logic; r1 : std_logic; r2 : std_logic; r3 : std_logic
) return std_logic is
begin
    case idx is
        when 0 => return r0;
        when 1 => return r1;
        when 2 => return r2;
        when others => return r3;
    end case;
end function;
function prior_at(
    idx : integer;
    p0 : prior_vec_t;
    p1 : prior_vec_t;
    p2 : prior_vec_t;
    p3 : prior_vec_t
) return prior_vec_t is
begin
    case idx is
        when 0 => return p0;
        when 1 => return p1;
        when 2 => return p2;
        when others => return p3;
    end case;
end function;
begin
    S_assign_valid_0      <= assign_valid_0_reg;
    V_assign_cpu_0        <= assign_cpu_0_reg;
    S_assign_interrupt_0  <= assign_interrupt_0_reg;
    S_assign_valid_1      <= assign_valid_1_reg;
    V_assign_cpu_1        <= assign_cpu_1_reg;
    S_assign_interrupt_1  <= assign_interrupt_1_reg;
    S_assign_valid_2      <= assign_valid_2_reg;
    V_assign_cpu_2        <= assign_cpu_2_reg;
    S_assign_interrupt_2  <= assign_interrupt_2_reg;
    S_assign_valid_3      <= assign_valid_3_reg;
    V_assign_cpu_3        <= assign_cpu_3_reg;
    S_assign_interrupt_3  <= assign_interrupt_3_reg;
    process (
        rr_pointer,
        S_top_valid_0, V_top_prior_0, S_top_valid_1, V_top_prior_1,
        S_top_valid_2, V_top_prior_2, S_top_valid_3, V_top_prior_3,
        S_cpu_busy_0, S_cpu_busy_1, S_cpu_busy_2, S_cpu_busy_3,
        S_cpu_running_0, S_cpu_running_1, S_cpu_running_2, S_cpu_running_3,
        V_cpu_prior_0, V_cpu_prior_1, V_cpu_prior_2, V_cpu_prior_3
    )
        variable used_cpu      : sl_array_t;
        variable task_valid    : sl_array_t;
        variable task_prior    : prior_vec_t;
        variable chosen_cpu    : integer;
        variable found         : boolean;
        variable idx           : integer;
        variable candidate_cpu : integer;
        variable candidate_set  : boolean;
        variable candidate_prio : prior_vec_t;
        variable last_selected : integer range 0 to 3;
    begin
        assign_valid_0_reg <= '0';
        assign_valid_1_reg <= '0';
        assign_valid_2_reg <= '0';
        assign_valid_3_reg <= '0';
        assign_cpu_0_reg   <= 0;
        assign_cpu_1_reg   <= 0;

```

```

assign_cpu_2_reg      <= 0;
assign_cpu_3_reg      <= 0;
assign_interrupt_0_reg <= '0';
assign_interrupt_1_reg <= '0';
assign_interrupt_2_reg <= '0';
assign_interrupt_3_reg <= '0';
last_cpu_next         <= rr_pointer;
used_cpu := (others => '0');
task_valid(0) := S_top_valid_0;
task_valid(1) := S_top_valid_1;
task_valid(2) := S_top_valid_2;
task_valid(3) := S_top_valid_3;
last_selected := rr_pointer;
for task_idx in 0 to 3 loop
    if task_idx = 0 then
        task_prior := V_top_prior_0;
    elsif task_idx = 1 then
        task_prior := V_top_prior_1;
    elsif task_idx = 2 then
        task_prior := V_top_prior_2;
    else
        task_prior := V_top_prior_3;
    end if;
    chosen_cpu := 0;
    found := false;
    if task_valid(task_idx) = '1' then
        for step in 1 to 4 loop
            idx := (last_selected + step) mod 4;
            if busy_at(idx, S_cpu_busy_0, S_cpu_busy_1, S_cpu_busy_2,
S_cpu_busy_3) = '0' and
                used_cpu(idx) = '0' and not found then
                chosen_cpu := idx;
                found := true;
            end if;
        end loop;
        if found then
            used_cpu(chosen_cpu) := '1';
            last_selected := chosen_cpu;
            case task_idx is
                when 0 =>
                    assign_valid_0_reg <= '1';
                    assign_cpu_0_reg <= chosen_cpu;
                    assign_interrupt_0_reg <= '0';
                when 1 =>
                    assign_valid_1_reg <= '1';
                    assign_cpu_1_reg <= chosen_cpu;
                    assign_interrupt_1_reg <= '0';
                when 2 =>
                    assign_valid_2_reg <= '1';
                    assign_cpu_2_reg <= chosen_cpu;
                    assign_interrupt_2_reg <= '0';
                when others =>
                    assign_valid_3_reg <= '1';
                    assign_cpu_3_reg <= chosen_cpu;
                    assign_interrupt_3_reg <= '0';
            end case;
        else
            candidate_cpu := 0;
            candidate_set := false;
            candidate_prio := (others => '0');
            for step in 1 to 4 loop
                idx := (last_selected + step) mod 4;

```

```

        if running_at(idx, S_cpu_running_0, S_cpu_running_1,
S_cpu_running_2, S_cpu_running_3) = '1' and
            used_cpu(idx) = '0' and
            unsigned(task_prior) < unsigned(prior_at(idx,
V_cpu_prior_0, V_cpu_prior_1, V_cpu_prior_2, V_cpu_prior_3)) then
                if not candidate_set or
                    unsigned(prior_at(idx, V_cpu_prior_0, V_cpu_prior_1,
V_cpu_prior_2, V_cpu_prior_3)) > unsigned(candidate_prio) then
                        candidate_cpu := idx;
                        candidate_prio := prior_at(idx, V_cpu_prior_0,
V_cpu_prior_1, V_cpu_prior_2, V_cpu_prior_3);
                        candidate_set := true;
                    end if;
                end if;
            end loop;
        if candidate_set then
            used_cpu(candidate_cpu) := '1';
            last_selected := candidate_cpu;
            case task_idx is
                when 0 =>
                    assign_valid_0_reg <= '1';
                    assign_cpu_0_reg <= candidate_cpu;
                    assign_interrupt_0_reg <= '1';
                when 1 =>
                    assign_valid_1_reg <= '1';
                    assign_cpu_1_reg <= candidate_cpu;
                    assign_interrupt_1_reg <= '1';
                when 2 =>
                    assign_valid_2_reg <= '1';
                    assign_cpu_2_reg <= candidate_cpu;
                    assign_interrupt_2_reg <= '1';
                when others =>
                    assign_valid_3_reg <= '1';
                    assign_cpu_3_reg <= candidate_cpu;
                    assign_interrupt_3_reg <= '1';
            end case;
        end if;
    end if;
end if;
end loop;
last_cpu_next <= last_selected;
end process;
process (CLK)
begin
    if rising_edge(CLK) then
        if reset = '1' then
            rr_pointer <= 3;
        else
            rr_pointer <= last_cpu_next;
        end if;
    end if;
end process;
end architecture behavioral;
block_synchron_naznach_global_abs.vhd
library ieee;
use ieee.std_logic_1164.all;
entity block_synchron_naznach_global_abs is
    port (
        CLK                : in  std_logic;
        reset               : in  std_logic;
        S_top_valid_0       : in  std_logic;
        V_top_task_id_0     : in  std_logic_vector(15 downto 0);
        V_top_prior_0       : in  std_logic_vector(3 downto 0);
    );
end entity;

```

```

V_top_wait_time_0      : in  integer;
V_top_system_time_0    : in  integer;
V_top_rem_time_0       : in  integer;
V_top_started_0        : in  std_logic;
S_top_valid_1          : in  std_logic;
V_top_task_id_1        : in  std_logic_vector(15 downto 0);
V_top_prior_1          : in  std_logic_vector(3 downto 0);
V_top_wait_time_1      : in  integer;
V_top_system_time_1    : in  integer;
V_top_rem_time_1       : in  integer;
V_top_started_1        : in  std_logic;
S_top_valid_2          : in  std_logic;
V_top_task_id_2        : in  std_logic_vector(15 downto 0);
V_top_prior_2          : in  std_logic_vector(3 downto 0);
V_top_wait_time_2      : in  integer;
V_top_system_time_2    : in  integer;
V_top_rem_time_2       : in  integer;
V_top_started_2        : in  std_logic;
S_top_valid_3          : in  std_logic;
V_top_task_id_3        : in  std_logic_vector(15 downto 0);
V_top_prior_3          : in  std_logic_vector(3 downto 0);
V_top_wait_time_3      : in  integer;
V_top_system_time_3    : in  integer;
V_top_rem_time_3       : in  integer;
V_top_started_3        : in  std_logic;
S_assign_valid_0        : in  std_logic;
V_assign_cpu_0         : in  integer range 0 to 3;
S_assign_interrupt_0    : in  std_logic;
S_assign_valid_1        : in  std_logic;
V_assign_cpu_1         : in  integer range 0 to 3;
S_assign_interrupt_1    : in  std_logic;
S_assign_valid_2        : in  std_logic;
V_assign_cpu_2         : in  integer range 0 to 3;
S_assign_interrupt_2    : in  std_logic;
S_assign_valid_3        : in  std_logic;
V_assign_cpu_3         : in  integer range 0 to 3;
S_assign_interrupt_3    : in  std_logic;
S_pop_top_0            : out std_logic;
S_pop_top_1            : out std_logic;
S_pop_top_2            : out std_logic;
S_pop_top_3            : out std_logic;
S_load_cpu_0           : out std_logic;
S_interrupt_cpu_0       : out std_logic;
V_task_id_to_cpu_0      : out std_logic_vector(15 downto 0);
V_prior_to_cpu_0        : out std_logic_vector(3 downto 0);
V_wait_to_cpu_0         : out integer;
V_system_to_cpu_0       : out integer;
V_rem_to_cpu_0          : out integer;
V_started_to_cpu_0      : out std_logic;
S_load_cpu_1           : out std_logic;
S_interrupt_cpu_1       : out std_logic;
V_task_id_to_cpu_1      : out std_logic_vector(15 downto 0);
V_prior_to_cpu_1        : out std_logic_vector(3 downto 0);
V_wait_to_cpu_1         : out integer;
V_system_to_cpu_1       : out integer;
V_rem_to_cpu_1          : out integer;
V_started_to_cpu_1      : out std_logic;
S_load_cpu_2           : out std_logic;
S_interrupt_cpu_2       : out std_logic;
V_task_id_to_cpu_2      : out std_logic_vector(15 downto 0);
V_prior_to_cpu_2        : out std_logic_vector(3 downto 0);
V_wait_to_cpu_2         : out integer;
V_system_to_cpu_2       : out integer;

```

```

V_rem_to_cpu_2      : out integer;
V_started_to_cpu_2  : out std_logic;
S_load_cpu_3        : out std_logic;
S_interrupt_cpu_3   : out std_logic;
V_task_id_to_cpu_3  : out std_logic_vector(15 downto 0);
V_prior_to_cpu_3    : out std_logic_vector(3 downto 0);
V_wait_to_cpu_3     : out integer;
V_system_to_cpu_3   : out integer;
V_rem_to_cpu_3      : out integer;
V_started_to_cpu_3  : out std_logic;
S_task_dispatched_0  : out std_logic;
V_dispatched_prior_0 : out std_logic_vector(3 downto 0);
S_task_dispatched_1  : out std_logic;
V_dispatched_prior_1 : out std_logic_vector(3 downto 0);
S_task_dispatched_2  : out std_logic;
V_dispatched_prior_2 : out std_logic_vector(3 downto 0);
S_task_dispatched_3  : out std_logic;
V_dispatched_prior_3 : out std_logic_vector(3 downto 0)
);
end entity block_synchron_naznach_global_abs;
architecture behavioral of block_synchron_naznach_global_abs is
  procedure clear_cpu(
    signal load_out      : out std_logic;
    signal intr_out      : out std_logic;
    signal task_id_out   : out std_logic_vector(15 downto 0);
    signal prior_out     : out std_logic_vector(3 downto 0);
    signal wait_out      : out integer;
    signal system_out    : out integer;
    signal rem_out       : out integer;
    signal started_out   : out std_logic
  ) is
  begin
    load_out      <= '0';
    intr_out      <= '0';
    task_id_out   <= (others => '0');
    prior_out     <= (others => '0');
    wait_out      <= 0;
    system_out    <= 0;
    rem_out       <= 0;
    started_out   <= '0';
  end procedure;
  procedure drive_cpu(
    constant cpu_idx      : in integer;
    constant intr_in      : in std_logic;
    constant task_id_in   : in std_logic_vector(15 downto 0);
    constant prior_in     : in std_logic_vector(3 downto 0);
    constant wait_in      : in integer;
    constant system_in    : in integer;
    constant rem_in       : in integer;
    constant started_in   : in std_logic;
    signal load_0         : out std_logic;
    signal intr_0         : out std_logic;
    signal task_id_0      : out std_logic_vector(15 downto 0);
    signal prior_0        : out std_logic_vector(3 downto 0);
    signal wait_0         : out integer;
    signal system_0       : out integer;
    signal rem_0          : out integer;
    signal started_0      : out std_logic;
    signal load_1         : out std_logic;
    signal intr_1         : out std_logic;
    signal task_id_1      : out std_logic_vector(15 downto 0);
    signal prior_1        : out std_logic_vector(3 downto 0);
    signal wait_1         : out integer;

```

```

signal system_1      : out integer;
signal rem_1         : out integer;
signal started_1     : out std_logic;
signal load_2        : out std_logic;
signal intr_2        : out std_logic;
signal task_id_2     : out std_logic_vector(15 downto 0);
signal prior_2       : out std_logic_vector(3 downto 0);
signal wait_2        : out integer;
signal system_2      : out integer;
signal rem_2         : out integer;
signal started_2     : out std_logic;
signal load_3        : out std_logic;
signal intr_3        : out std_logic;
signal task_id_3     : out std_logic_vector(15 downto 0);
signal prior_3       : out std_logic_vector(3 downto 0);
signal wait_3        : out integer;
signal system_3      : out integer;
signal rem_3         : out integer;
signal started_3     : out std_logic;
) is
begin
  case cpu_idx is
    when 0 =>
      load_0 <= '1';
      intr_0 <= intr_in;
      task_id_0 <= task_id_in;
      prior_0 <= prior_in;
      if wait_in <= 0 then
        wait_0 <= 1;
      else
        wait_0 <= wait_in;
      end if;
      system_0 <= system_in;
      rem_0 <= rem_in;
      started_0 <= started_in;
    when 1 =>
      load_1 <= '1';
      intr_1 <= intr_in;
      task_id_1 <= task_id_in;
      prior_1 <= prior_in;
      if wait_in <= 0 then
        wait_1 <= 1;
      else
        wait_1 <= wait_in;
      end if;
      system_1 <= system_in;
      rem_1 <= rem_in;
      started_1 <= started_in;
    when 2 =>
      load_2 <= '1';
      intr_2 <= intr_in;
      task_id_2 <= task_id_in;
      prior_2 <= prior_in;
      if wait_in <= 0 then
        wait_2 <= 1;
      else
        wait_2 <= wait_in;
      end if;
      system_2 <= system_in;
      rem_2 <= rem_in;
      started_2 <= started_in;
    when others =>
      load_3 <= '1';
  end case;
end;

```

```

        intr_3 <= intr_in;
        task_id_3 <= task_id_in;
        prior_3 <= prior_in;
        if wait_in <= 0 then
            wait_3 <= 1;
        else
            wait_3 <= wait_in;
        end if;
        system_3 <= system_in;
        rem_3 <= rem_in;
        started_3 <= started_in;
    end case;
end procedure;
begin
    process (
        reset,
        S_top_valid_0, V_top_task_id_0, V_top_prior_0, V_top_wait_time_0,
        V_top_system_time_0, V_top_rem_time_0, V_top_started_0,
        S_top_valid_1, V_top_task_id_1, V_top_prior_1, V_top_wait_time_1,
        V_top_system_time_1, V_top_rem_time_1, V_top_started_1,
        S_top_valid_2, V_top_task_id_2, V_top_prior_2, V_top_wait_time_2,
        V_top_system_time_2, V_top_rem_time_2, V_top_started_2,
        S_top_valid_3, V_top_task_id_3, V_top_prior_3, V_top_wait_time_3,
        V_top_system_time_3, V_top_rem_time_3, V_top_started_3,
        S_assign_valid_0, V_assign_cpu_0, S_assign_interrupt_0,
        S_assign_valid_1, V_assign_cpu_1, S_assign_interrupt_1,
        S_assign_valid_2, V_assign_cpu_2, S_assign_interrupt_2,
        S_assign_valid_3, V_assign_cpu_3, S_assign_interrupt_3
    )
    begin
        S_pop_top_0 <= '0';
        S_pop_top_1 <= '0';
        S_pop_top_2 <= '0';
        S_pop_top_3 <= '0';
        clear_cpu(S_load_cpu_0,          S_interrupt_cpu_0,          V_task_id_to_cpu_0,
V_prior_to_cpu_0,          V_wait_to_cpu_0,          V_system_to_cpu_0,          V_rem_to_cpu_0,
V_started_to_cpu_0);
        clear_cpu(S_load_cpu_1,          S_interrupt_cpu_1,          V_task_id_to_cpu_1,
V_prior_to_cpu_1,          V_wait_to_cpu_1,          V_system_to_cpu_1,          V_rem_to_cpu_1,
V_started_to_cpu_1);
        clear_cpu(S_load_cpu_2,          S_interrupt_cpu_2,          V_task_id_to_cpu_2,
V_prior_to_cpu_2,          V_wait_to_cpu_2,          V_system_to_cpu_2,          V_rem_to_cpu_2,
V_started_to_cpu_2);
        clear_cpu(S_load_cpu_3,          S_interrupt_cpu_3,          V_task_id_to_cpu_3,
V_prior_to_cpu_3,          V_wait_to_cpu_3,          V_system_to_cpu_3,          V_rem_to_cpu_3,
V_started_to_cpu_3);
        S_task_dispatched_0 <= '0';
        S_task_dispatched_1 <= '0';
        S_task_dispatched_2 <= '0';
        S_task_dispatched_3 <= '0';
        V_dispatched_prior_0 <= (others => '0');
        V_dispatched_prior_1 <= (others => '0');
        V_dispatched_prior_2 <= (others => '0');
        V_dispatched_prior_3 <= (others => '0');
        if reset = '0' then
            if S_top_valid_0 = '1' and S_assign_valid_0 = '1' then
                S_pop_top_0 <= '1';
                S_task_dispatched_0 <= '1';
                V_dispatched_prior_0 <= V_top_prior_0;
            end if;
        end if;
    end
end process;

```

```

        drive_cpu(V_assign_cpu_0, S_assign_interrupt_0, V_top_task_id_0,
V_top_prior_0,
        V_top_wait_time_0, V_top_system_time_0, V_top_rem_time_0,
V_top_started_0,
        S_load_cpu_0, S_interrupt_cpu_0, V_task_id_to_cpu_0,
V_prior_to_cpu_0,
        V_wait_to_cpu_0, V_system_to_cpu_0, V_rem_to_cpu_0,
V_started_to_cpu_0,
        S_load_cpu_1, S_interrupt_cpu_1, V_task_id_to_cpu_1,
V_prior_to_cpu_1,
        V_wait_to_cpu_1, V_system_to_cpu_1, V_rem_to_cpu_1,
V_started_to_cpu_1,
        S_load_cpu_2, S_interrupt_cpu_2, V_task_id_to_cpu_2,
V_prior_to_cpu_2,
        V_wait_to_cpu_2, V_system_to_cpu_2, V_rem_to_cpu_2,
V_started_to_cpu_2,
        S_load_cpu_3, S_interrupt_cpu_3, V_task_id_to_cpu_3,
V_prior_to_cpu_3,
        V_wait_to_cpu_3, V_system_to_cpu_3, V_rem_to_cpu_3,
V_started_to_cpu_3);
    end if;
    if S_top_valid_1 = '1' and S_assign_valid_1 = '1' then
        S_pop_top_1 <= '1';
        S_task_dispatched_1 <= '1';
        V_dispatched_prior_1 <= V_top_prior_1;
        drive_cpu(V_assign_cpu_1, S_assign_interrupt_1, V_top_task_id_1,
V_top_prior_1,
        V_top_wait_time_1, V_top_system_time_1, V_top_rem_time_1,
V_top_started_1,
        S_load_cpu_0, S_interrupt_cpu_0, V_task_id_to_cpu_0,
V_prior_to_cpu_0,
        V_wait_to_cpu_0, V_system_to_cpu_0, V_rem_to_cpu_0,
V_started_to_cpu_0,
        S_load_cpu_1, S_interrupt_cpu_1, V_task_id_to_cpu_1,
V_prior_to_cpu_1,
        V_wait_to_cpu_1, V_system_to_cpu_1, V_rem_to_cpu_1,
V_started_to_cpu_1,
        S_load_cpu_2, S_interrupt_cpu_2, V_task_id_to_cpu_2,
V_prior_to_cpu_2,
        V_wait_to_cpu_2, V_system_to_cpu_2, V_rem_to_cpu_2,
V_started_to_cpu_2,
        S_load_cpu_3, S_interrupt_cpu_3, V_task_id_to_cpu_3,
V_prior_to_cpu_3,
        V_wait_to_cpu_3, V_system_to_cpu_3, V_rem_to_cpu_3,
V_started_to_cpu_3);
    end if;
    if S_top_valid_2 = '1' and S_assign_valid_2 = '1' then
        S_pop_top_2 <= '1';
        S_task_dispatched_2 <= '1';
        V_dispatched_prior_2 <= V_top_prior_2;
        drive_cpu(V_assign_cpu_2, S_assign_interrupt_2, V_top_task_id_2,
V_top_prior_2,
        V_top_wait_time_2, V_top_system_time_2, V_top_rem_time_2,
V_top_started_2,
        S_load_cpu_0, S_interrupt_cpu_0, V_task_id_to_cpu_0,
V_prior_to_cpu_0,
        V_wait_to_cpu_0, V_system_to_cpu_0, V_rem_to_cpu_0,
V_started_to_cpu_0,
        S_load_cpu_1, S_interrupt_cpu_1, V_task_id_to_cpu_1,
V_prior_to_cpu_1,
        V_wait_to_cpu_1, V_system_to_cpu_1, V_rem_to_cpu_1,
V_started_to_cpu_1,

```

```

V_prior_to_cpu_2,      S_load_cpu_2,      S_interrupt_cpu_2,      V_task_id_to_cpu_2,
V_started_to_cpu_2,    V_wait_to_cpu_2,      V_system_to_cpu_2,      V_rem_to_cpu_2,
V_prior_to_cpu_3,      S_load_cpu_3,      S_interrupt_cpu_3,      V_task_id_to_cpu_3,
V_started_to_cpu_3);
V_wait_to_cpu_3,      V_system_to_cpu_3,      V_rem_to_cpu_3,
end if;
if S_top_valid_3 = '1' and S_assign_valid_3 = '1' then
    S_pop_top_3 <= '1';
    S_task_dispatched_3 <= '1';
    V_dispatched_prior_3 <= V_top_prior_3;
    drive_cpu(V_assign_cpu_3, S_assign_interrupt_3, V_top_task_id_3,
V_top_prior_3,
V_top_wait_time_3, V_top_system_time_3, V_top_rem_time_3,
V_top_started_3,
S_load_cpu_0, S_interrupt_cpu_0, V_task_id_to_cpu_0,
V_prior_to_cpu_0,
V_wait_to_cpu_0, V_system_to_cpu_0, V_rem_to_cpu_0,
V_started_to_cpu_0,
S_load_cpu_1, S_interrupt_cpu_1, V_task_id_to_cpu_1,
V_prior_to_cpu_1,
V_wait_to_cpu_1, V_system_to_cpu_1, V_rem_to_cpu_1,
V_started_to_cpu_1,
S_load_cpu_2, S_interrupt_cpu_2, V_task_id_to_cpu_2,
V_prior_to_cpu_2,
V_wait_to_cpu_2, V_system_to_cpu_2, V_rem_to_cpu_2,
V_started_to_cpu_2,
S_load_cpu_3, S_interrupt_cpu_3, V_task_id_to_cpu_3,
V_prior_to_cpu_3,
V_wait_to_cpu_3, V_system_to_cpu_3, V_rem_to_cpu_3,
V_started_to_cpu_3);
end if;
end if;
end process;
end architecture behavioral;
proc_abs.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
entity proc_abs is
    generic (
        MU_C : real := 1.0 / 9.5;
        PROCESSOR_SEED1_C : positive := 301;
        PROCESSOR_SEED2_C : positive := 709
    );
    port (
        CLK : in std_logic;
        reset : in std_logic;
        S_load_task : in std_logic;
        S_interrupt : in std_logic;
        V_task_id_in : in std_logic_vector(15 downto 0);
        V_prior_in : in std_logic_vector(3 downto 0);
        V_wait_time_in : in integer;
        V_system_time_in : in integer;
        V_rem_time_in : in integer;
        V_started_in : in std_logic;
        S_cpu_busy : out std_logic;
        S_cpu_running : out std_logic;
        S_cpu_dispatching : out std_logic;
        V_current_task_id : out std_logic_vector(15 downto 0);

```

```

        V_current_prior      : out std_logic_vector(3 downto 0);
        V_current_rem_time   : out integer;
        S_preempted_valid    : out std_logic;
        V_preempted_task_id  : out std_logic_vector(15 downto 0);
        V_preempted_prior    : out std_logic_vector(3 downto 0);
        V_preempted_wait     : out integer;
        V_preempted_system   : out integer;
        V_preempted_rem      : out integer;
        V_preempted_started  : out std_logic;
        S_task_done          : out std_logic;
        V_done_task_id       : out std_logic_vector(15 downto 0);
        V_done_prior         : out std_logic_vector(3 downto 0);
        V_done_wait          : out integer;
        V_done_system        : out integer;
        S_useful_busy_tick   : out std_logic
    );
end entity proc_abs;
architecture behavioral of proc_abs is
    type cpu_state_t is (FREE, RUNNING);
    signal state_reg      : cpu_state_t := FREE;
    signal task_id_reg    : std_logic_vector(15 downto 0) := (others => '0');
    signal prior_reg      : std_logic_vector(3 downto 0) := (others => '0');
    signal wait_reg       : integer := 0;
    signal system_reg     : integer := 0;
    signal rem_reg        : integer := 0;
    signal preempted_valid_reg : std_logic := '0';
    signal preempted_task_id_reg : std_logic_vector(15 downto 0) := (others => '0');
    signal preempted_prior_reg : std_logic_vector(3 downto 0) := (others => '0');
    signal preempted_wait_reg : integer := 0;
    signal preempted_system_reg : integer := 0;
    signal preempted_rem_reg : integer := 0;
    signal task_done_reg   : std_logic := '0';
    signal done_task_id_reg : std_logic_vector(15 downto 0) := (others => '0');
    signal done_prior_reg  : std_logic_vector(3 downto 0) := (others => '0');
    signal done_wait_reg   : integer := 0;
    signal done_system_reg : integer := 0;
    signal useful_tick_reg : std_logic := '0';
begin
    S_cpu_busy      <= '1' when state_reg /= FREE else '0';
    S_cpu_running   <= '1' when state_reg = RUNNING else '0';
    S_cpu_dispatching <= '0';
    V_current_task_id <= task_id_reg;
    V_current_prior   <= prior_reg;
    V_current_rem_time <= rem_reg;
    S_preempted_valid <= preempted_valid_reg;
    V_preempted_task_id <= preempted_task_id_reg;
    V_preempted_prior <= preempted_prior_reg;
    V_preempted_wait <= preempted_wait_reg;
    V_preempted_system <= preempted_system_reg;
    V_preempted_rem <= preempted_rem_reg;
    V_preempted_started <= '1';
    S_task_done <= task_done_reg;
    V_done_task_id <= done_task_id_reg;
    V_done_prior <= done_prior_reg;
    V_done_wait <= done_wait_reg;
    V_done_system <= done_system_reg;
    S_useful_busy_tick <= useful_tick_reg;
    process (CLK)
        variable seed1      : positive := PROCESSOR_SEED1_C;
        variable seed2      : positive := PROCESSOR_SEED2_C;
        variable rand        : real;
        variable service_real : real;
        variable service_ticks : integer;

```

```

variable next_rem      : integer;
begin
  if rising_edge(CLK) then
    preempted_valid_reg <= '0';
    task_done_reg       <= '0';
    useful_tick_reg     <= '0';
    if reset = '1' then
      seed1 := PROCESSOR_SEED1_C;
      seed2 := PROCESSOR_SEED2_C;
      state_reg <= FREE;
      task_id_reg <= (others => '0');
      prior_reg <= (others => '0');
      wait_reg <= 0;
      system_reg <= 0;
      rem_reg <= 0;
    elsif S_load_task = '1' then
      if S_interrupt = '1' and state_reg = RUNNING then
        preempted_valid_reg <= '1';
        preempted_task_id_reg <= task_id_reg;
        preempted_prior_reg <= prior_reg;
        preempted_wait_reg <= wait_reg;
        preempted_system_reg <= system_reg;
        preempted_rem_reg <= rem_reg;
      end if;
      if V_started_in = '1' and V_rem_time_in > 0 then
        service_ticks := V_rem_time_in;
      else
        uniform(seed1, seed2, rand);
        service_real := -1.0 / MU_C * log(1.0 - rand);
        service_ticks := integer(floor(service_real));
        if service_ticks <= 0 then
          service_ticks := 1;
        end if;
      end if;
      task_id_reg <= V_task_id_in;
      prior_reg <= V_prior_in;
      wait_reg <= V_wait_time_in;
      system_reg <= V_system_time_in;
      rem_reg <= service_ticks;
      state_reg <= RUNNING;
    else
      case state_reg is
        when FREE =>
          null;
        when RUNNING =>
          useful_tick_reg <= '1';
          system_reg <= system_reg + 1;
          next_rem := rem_reg - 1;
          rem_reg <= next_rem;
          if next_rem <= 0 then
            task_done_reg <= '1';
            done_task_id_reg <= task_id_reg;
            done_prior_reg <= prior_reg;
            done_wait_reg <= wait_reg;
            done_system_reg <= system_reg + 1;
            state_reg <= FREE;
            rem_reg <= 0;
          end if;
        end case;
      end if;
    end if;
  end process;
end architecture behavioral;

```

```

block_vibora_oher.vhd
library ieee;
use ieee.std_logic_1164.all;
entity block_vibora_oher is
    port (
        CLK                : in  std_logic;
        reset               : in  std_logic;
        S_task_arrived      : in  std_logic;
        V_use_1             : in  integer;
        V_use_2             : in  integer;
        V_use_3             : in  integer;
        V_use_4             : in  integer;
        S_prior_full_1      : in  std_logic;
        S_prior_full_2      : in  std_logic;
        S_prior_full_3      : in  std_logic;
        S_prior_full_4      : in  std_logic;
        S_oher_sel_1        : out std_logic;
        S_oher_sel_2        : out std_logic;
        S_oher_sel_3        : out std_logic;
        S_oher_sel_4        : out std_logic;
        V_selected_oher     : out integer;
        S_all_queues_full   : out std_logic
    );
end entity block_vibora_oher;
architecture behavioral of block_vibora_oher is
    signal sel_reg          : integer range -1 to 3 := -1;
    signal last_sel_reg     : integer range 0 to 3 := 3;
    signal all_full_reg     : std_logic := '0';
    function full_at(
        idx : integer;
        f0  : std_logic; f1 : std_logic; f2 : std_logic; f3 : std_logic
    ) return std_logic is
    begin
        case idx is
            when 0 => return f0;
            when 1 => return f1;
            when 2 => return f2;
            when others => return f3;
        end case;
    end function;
    function use_at(
        idx : integer;
        u0  : integer; u1 : integer; u2 : integer; u3 : integer
    ) return integer is
    begin
        case idx is
            when 0 => return u0;
            when 1 => return u1;
            when 2 => return u2;
            when others => return u3;
        end case;
    end function;
begin
    S_oher_sel_1 <= '1' when sel_reg = 0 else '0';
    S_oher_sel_2 <= '1' when sel_reg = 1 else '0';
    S_oher_sel_3 <= '1' when sel_reg = 2 else '0';
    S_oher_sel_4 <= '1' when sel_reg = 3 else '0';
    V_selected_oher <= sel_reg;
    S_all_queues_full <= all_full_reg;
    process (CLK)
        variable all_full : boolean;
        variable min_use  : integer;
        variable idx      : integer;

```

```

        variable selected : integer;
begin
    if rising_edge(CLK) then
        sel_reg <= -1;
        all_full_reg <= '0';
        if reset = '1' then
            sel_reg <= -1;
            last_sel_reg <= 3;
            all_full_reg <= '0';
        elsif S_task_arrived = '1' then
            all_full := (S_prior_full_1 = '1') and (S_prior_full_2 = '1') and
                        (S_prior_full_3 = '1') and (S_prior_full_4 = '1');
            if all_full then
                all_full_reg <= '1';
                sel_reg <= -1;
            else
                min_use := 999999;
                for step in 1 to 4 loop
                    idx := (last_sel_reg + step) mod 4;
                    if full_at(idx, S_prior_full_1, S_prior_full_2,
S_prior_full_3, S_prior_full_4) = '0' then
                        if use_at(idx, V_use_1, V_use_2, V_use_3, V_use_4) <
min_use then
                            min_use := use_at(idx, V_use_1, V_use_2, V_use_3,
V_use_4);
                        end if;
                    end if;
                end loop;
                selected := -1;
                for step in 1 to 4 loop
                    idx := (last_sel_reg + step) mod 4;
                    if selected = -1 and
                        full_at(idx, S_prior_full_1, S_prior_full_2,
S_prior_full_3, S_prior_full_4) = '0' and
                        use_at(idx, V_use_1, V_use_2, V_use_3, V_use_4) = min_use
then
                            selected := idx;
                        end if;
                    end loop;
                sel_reg <= selected;
                if selected >= 0 then
                    last_sel_reg <= selected;
                end if;
            end if;
        end if;
    end if;
end process;
end architecture behavioral;
block_nazn_zadach.vhd
library ieee;
use ieee.std_logic_1164.all;
entity block_nazn_zadach is
    port (
        CLK                : in  std_logic;
        reset               : in  std_logic;
        S_task_arrived      : in  std_logic;
        V_task_id           : in  std_logic_vector(15 downto 0);
        V_task_prior        : in  std_logic_vector(3 downto 0);
        S_oher_sel_1        : in  std_logic;
        S_oher_sel_2        : in  std_logic;
        S_oher_sel_3        : in  std_logic;
        S_oher_sel_4        : in  std_logic;
        S_all_queues_full   : in  std_logic;

```

```

        S_prior_full_1      : in  std_logic;
        S_prior_full_2      : in  std_logic;
        S_prior_full_3      : in  std_logic;
        S_prior_full_4      : in  std_logic;
        V_task_id_o          : out std_logic_vector(15 downto 0);
        V_task_prior_o       : out std_logic_vector(3  downto 0);
        S_wrreq_1            : out std_logic;
        S_wrreq_2            : out std_logic;
        S_wrreq_3            : out std_logic;
        S_wrreq_4            : out std_logic;
        S_task_created       : out std_logic;
        V_created_prior      : out std_logic_vector(3  downto 0);
        S_task_refused       : out std_logic;
        V_refused_prior      : out std_logic_vector(3  downto 0)
    );
end entity block_nazn_zadach;
architecture behavioral of block_nazn_zadach is
    signal valid_d1          : std_logic := '0';
    signal task_id_d1        : std_logic_vector(15 downto 0) := (others => '0');
    signal task_prior_d1     : std_logic_vector(3  downto 0) := (others => '0');
    signal task_id_reg       : std_logic_vector(15 downto 0) := (others => '0');
    signal task_prior_reg    : std_logic_vector(3  downto 0) := (others => '0');
    signal wrreq_1_reg       : std_logic := '0';
    signal wrreq_2_reg       : std_logic := '0';
    signal wrreq_3_reg       : std_logic := '0';
    signal wrreq_4_reg       : std_logic := '0';
    signal created_reg        : std_logic := '0';
    signal created_prior_reg  : std_logic_vector(3  downto 0) := (others => '0');
    signal refused_reg        : std_logic := '0';
    signal refused_prior_reg  : std_logic_vector(3  downto 0) := (others => '0');
begin
    V_task_id_o <= task_id_reg;
    V_task_prior_o <= task_prior_reg;
    S_wrreq_1 <= wrreq_1_reg;
    S_wrreq_2 <= wrreq_2_reg;
    S_wrreq_3 <= wrreq_3_reg;
    S_wrreq_4 <= wrreq_4_reg;
    S_task_created <= created_reg;
    V_created_prior <= created_prior_reg;
    S_task_refused <= refused_reg;
    V_refused_prior <= refused_prior_reg;
    process (CLK)
        variable can_write : std_logic;
    begin
        if rising_edge(CLK) then
            wrreq_1_reg <= '0';
            wrreq_2_reg <= '0';
            wrreq_3_reg <= '0';
            wrreq_4_reg <= '0';
            created_reg <= '0';
            refused_reg <= '0';
            if reset = '1' then
                valid_d1 <= '0';
                task_id_d1 <= (others => '0');
                task_prior_d1 <= (others => '0');
                task_id_reg <= (others => '0');
                task_prior_reg <= (others => '0');
                created_prior_reg <= (others => '0');
                refused_prior_reg <= (others => '0');
            else
                if valid_d1 = '1' then
                    task_id_reg <= task_id_d1;
                    task_prior_reg <= task_prior_d1;
                end if;
            end if;
        end if;
    end process;
end architecture;

```

```

        created_reg <= '1';
        created_prior_reg <= task_prior_d1;
        can_write := '0';
        if S_all_queues_full = '1' then
            refused_reg <= '1';
            refused_prior_reg <= task_prior_d1;
        elsif S_oher_sel_1 = '1' then
            if S_prior_full_1 = '0' then
                can_write := '1';
            end if;
            wrreq_1_reg <= can_write;
        elsif S_oher_sel_2 = '1' then
            if S_prior_full_2 = '0' then
                can_write := '1';
            end if;
            wrreq_2_reg <= can_write;
        elsif S_oher_sel_3 = '1' then
            if S_prior_full_3 = '0' then
                can_write := '1';
            end if;
            wrreq_3_reg <= can_write;
        elsif S_oher_sel_4 = '1' then
            if S_prior_full_4 = '0' then
                can_write := '1';
            end if;
            wrreq_4_reg <= can_write;
        else
            refused_reg <= '1';
            refused_prior_reg <= task_prior_d1;
        end if;
        if can_write = '0' and S_all_queues_full = '0' and
            (S_oher_sel_1 = '1' or S_oher_sel_2 = '1' or S_oher_sel_3
= '1' or S_oher_sel_4 = '1') then
            refused_reg <= '1';
            refused_prior_reg <= task_prior_d1;
        end if;
    end if;
    valid_d1 <= S_task_arrived;
    if S_task_arrived = '1' then
        task_id_d1 <= V_task_id;
        task_prior_d1 <= V_task_prior;
    end if;
end if;
end if;
end process;
end architecture behavioral;
block_oher_zad_prior_local.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity block_oher_zad_prior_local is
    generic (
        QUEUE_SIZE_C : integer := 4
    );
    port (
        CLK                : in  std_logic;
        reset               : in  std_logic;
        S_wrreq_preempted   : in  std_logic;
        V_preempted_task_id : in  std_logic_vector(15 downto 0);
        V_preempted_prior   : in  std_logic_vector(3 downto 0);
        V_preempted_wait_time : in  integer;
        V_preempted_system_time : in  integer;
        V_preempted_rem_time : in  integer;

```

```

V_preempted_started      : in  std_logic;
S_wrreq_new              : in  std_logic;
V_new_task_id            : in  std_logic_vector(15 downto 0);
V_new_prior              : in  std_logic_vector(3 downto 0);
S_pop_top                : in  std_logic;
S_prior_full             : out std_logic;
S_prior_empty            : out std_logic;
V_use                    : out integer;
S_top_valid              : out std_logic;
V_top_task_id            : out std_logic_vector(15 downto 0);
V_top_prior              : out std_logic_vector(3 downto 0);
V_top_wait_time          : out integer;
V_top_system_time        : out integer;
V_top_rem_time           : out integer;
V_top_started            : out std_logic
);
end entity block_oher_zad_prior_local;
architecture behavioral of block_oher_zad_prior_local is
    subtype queue_index_t is integer range 0 to 3;
    type slv16_array_t is array (queue_index_t) of std_logic_vector(15 downto 0);
    type slv4_array_t  is array (queue_index_t) of std_logic_vector(3 downto 0);
    type int_array_t   is array (queue_index_t) of integer;
    type sl_array_t    is array (queue_index_t) of std_logic;
    signal q_task_id : slv16_array_t := (others => (others => '0'));
    signal q_prior   : slv4_array_t  := (others => (others => '0'));
    signal q_wait    : int_array_t   := (others => 0);
    signal q_system  : int_array_t   := (others => 0);
    signal q_rem     : int_array_t   := (others => 0);
    signal q_started : sl_array_t    := (others => '0');
    signal q_count   : integer range 0 to 4 := 0;
    signal seq_next  : integer := 0;
    procedure insert_task_sorted(
        variable n_task_id : inout slv16_array_t;
        variable n_prior   : inout slv4_array_t;
        variable n_wait    : inout int_array_t;
        variable n_system  : inout int_array_t;
        variable n_rem     : inout int_array_t;
        variable n_started : inout sl_array_t;
        variable n_count   : inout integer;
        variable n_seq_next : inout integer;
        constant task_id_in : in std_logic_vector(15 downto 0);
        constant prior_in   : in std_logic_vector(3 downto 0);
        constant wait_in    : in integer;
        constant system_in  : in integer;
        constant rem_in     : in integer;
        constant started_in : in std_logic
    ) is
        variable insert_pos : integer;
    begin
        if n_count >= QUEUE_SIZE_C then
            return;
        end if;
        insert_pos := n_count;
        for i in 0 to 3 loop
            if i < n_count then
                if unsigned(n_prior(i)) > unsigned(prior_in) then
                    insert_pos := i;
                    exit;
                end if;
            end if;
        end loop;
        for i in 3 downto 1 loop
            if i > insert_pos and i <= n_count then

```

```

        n_task_id(i) := n_task_id(i - 1);
        n_prior(i)   := n_prior(i - 1);
        n_wait(i)    := n_wait(i - 1);
        n_system(i)  := n_system(i - 1);
        n_rem(i)     := n_rem(i - 1);
        n_started(i) := n_started(i - 1);
    end if;
end loop;
n_task_id(insert_pos) := task_id_in;
n_prior(insert_pos)   := prior_in;
n_wait(insert_pos)    := wait_in;
n_system(insert_pos)  := system_in;
n_rem(insert_pos)     := rem_in;
n_started(insert_pos) := started_in;
n_seq_next := n_seq_next + 1;
n_count := n_count + 1;
end procedure;
begin
    S_prior_full  <= '1' when q_count >= QUEUE_SIZE_C else '0';
    S_prior_empty <= '1' when q_count = 0 else '0';
    V_use         <= q_count;
    S_top_valid   <= '1' when q_count > 0 else '0';
    V_top_task_id <= q_task_id(0);
    V_top_prior   <= q_prior(0);
    V_top_wait_time <= q_wait(0);
    V_top_system_time <= q_system(0);
    V_top_rem_time <= q_rem(0);
    V_top_started <= q_started(0);
    process (CLK)
        variable n_task_id : slv16_array_t;
        variable n_prior   : slv4_array_t;
        variable n_wait    : int_array_t;
        variable n_system  : int_array_t;
        variable n_rem     : int_array_t;
        variable n_started : sl_array_t;
        variable n_count   : integer;
        variable n_seq_next : integer;
    begin
        if rising_edge(CLK) then
            if reset = '1' then
                q_task_id <= (others => (others => '0'));
                q_prior   <= (others => (others => '0'));
                q_wait    <= (others => 0);
                q_system  <= (others => 0);
                q_rem     <= (others => 0);
                q_started <= (others => '0');
                q_count   <= 0;
                seq_next  <= 0;
            else
                n_task_id := q_task_id;
                n_prior   := q_prior;
                n_wait    := q_wait;
                n_system  := q_system;
                n_rem     := q_rem;
                n_started := q_started;
                n_count   := q_count;
                n_seq_next := seq_next;
                if S_pop_top = '1' and n_count > 0 then
                    for i in 0 to 2 loop
                        n_task_id(i) := n_task_id(i + 1);
                        n_prior(i)   := n_prior(i + 1);
                        n_wait(i)    := n_wait(i + 1);
                        n_system(i)  := n_system(i + 1);
                    end loop;
                end if;
            end if;
        end if;
    end process;
end;

```

```

        n_rem(i)      := n_rem(i + 1);
        n_started(i) := n_started(i + 1);
    end loop;
    n_count := n_count - 1;
end if;
if S_wrreq_preempted = '1' and n_count < QUEUE_SIZE_C then
    insert_task_sorted(
        n_task_id, n_prior, n_wait, n_system, n_rem, n_started,
        n_count, n_seq_next,
        V_preempted_task_id, V_preempted_prior,
        V_preempted_wait_time, V_preempted_system_time,
        V_preempted_rem_time, V_preempted_started
    );
end if;
if S_wrreq_new = '1' and n_count < QUEUE_SIZE_C then
    insert_task_sorted(
        n_task_id, n_prior, n_wait, n_system, n_rem, n_started,
        n_count, n_seq_next,
        V_new_task_id, V_new_prior, 0, 0, 0, '0'
    );
end if;
for i in 0 to 3 loop
    if i < n_count then
        n_wait(i) := n_wait(i) + 1;
        n_system(i) := n_system(i) + 1;
    else
        n_task_id(i) := (others => '0');
        n_prior(i) := (others => '0');
        n_wait(i) := 0;
        n_system(i) := 0;
        n_rem(i) := 0;
        n_started(i) := '0';
    end if;
end loop;
q_task_id <= n_task_id;
q_prior <= n_prior;
q_wait <= n_wait;
q_system <= n_system;
q_rem <= n_rem;
q_started <= n_started;
q_count <= n_count;
seq_next <= n_seq_next;
end if;
end if;
end process;
end architecture behavioral;
block_naznach_local_proc.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity block_naznach_local_proc is
    port (
        reset                : in  std_logic;
        S_top_valid          : in  std_logic;
        V_top_task_id        : in  std_logic_vector(15 downto 0);
        V_top_prior          : in  std_logic_vector(3 downto 0);
        V_top_wait_time      : in  integer;
        V_top_system_time    : in  integer;
        V_top_rem_time       : in  integer;
        V_top_started        : in  std_logic;
        S_cpu_busy            : in  std_logic;
        S_cpu_running        : in  std_logic;
        V_cpu_prior          : in  std_logic_vector(3 downto 0);
    );
end entity;

```

```

        S_pop_top           : out std_logic;
        S_load_cpu          : out std_logic;
        S_interrupt_cpu     : out std_logic;
        V_task_id_to_cpu    : out std_logic_vector(15 downto 0);
        V_prior_to_cpu      : out std_logic_vector(3 downto 0);
        V_wait_to_cpu       : out integer;
        V_system_to_cpu     : out integer;
        V_rem_to_cpu        : out integer;
        V_started_to_cpu    : out std_logic;
        S_task_dispatched    : out std_logic;
        V_dispatched_prior   : out std_logic_vector(3 downto 0)
    );
end entity block_naznach_local_proc;
architecture behavioral of block_naznach_local_proc is
begin
    process (
        reset, S_top_valid, V_top_task_id, V_top_prior, V_top_wait_time,
        V_top_system_time, V_top_rem_time, V_top_started, S_cpu_busy,
        S_cpu_running, V_cpu_prior
    )
        variable do_load : std_logic;
        variable do_intr : std_logic;
    begin
        S_pop_top <= '0';
        S_load_cpu <= '0';
        S_interrupt_cpu <= '0';
        S_task_dispatched <= '0';
        V_dispatched_prior <= (others => '0');
        V_task_id_to_cpu <= V_top_task_id;
        V_prior_to_cpu <= V_top_prior;
        if V_top_wait_time <= 0 then
            V_wait_to_cpu <= 1;
        else
            V_wait_to_cpu <= V_top_wait_time;
        end if;
        V_system_to_cpu <= V_top_system_time;
        V_rem_to_cpu <= V_top_rem_time;
        V_started_to_cpu <= V_top_started;
        do_load := '0';
        do_intr := '0';
        if reset = '0' and S_top_valid = '1' then
            if S_cpu_busy = '0' then
                do_load := '1';
            elsif S_cpu_running = '1' and unsigned(V_top_prior) <
unsigned(V_cpu_prior) then
                do_load := '1';
                do_intr := '1';
            end if;
        end if;
        if do_load = '1' then
            S_pop_top <= '1';
            S_load_cpu <= '1';
            S_interrupt_cpu <= do_intr;
            S_task_dispatched <= '1';
            V_dispatched_prior <= V_top_prior;
        end if;
    end process;
end architecture behavioral;

```

Приложение В

(Обязательное)

Программа имитационного моделирования многопроцессорных приоритетных
планировщиков задач

Презентация

					ПГУ 1.090501.05.001 ПЗ	Лист
						148
Изм.	Лист	№ докум.	Подпись	Дата		

ДИПЛОМНЫЙ ПРОЕКТ

Программа имитационного моделирования многопроцессорных приоритетных планировщиков задач

Цель работы – разработка моделей многопроцессорных приоритетных планировщиков задач и имитационное моделирование их работы для получения и сравнения временных характеристик с использованием языка VHDL.

Выполнил: Тишаев Максим Русланович, гр. 21ВОЭ1
Руководитель: Бикташев Р. А.
Пенза, 2026

1

Актуальность работы



Почему задача важна

многопроцессорные системы требуют распределения задач между процессорами
выбор очереди и режима обслуживания влияет на время отклика
аппаратное моделирование позволяет оценить поведение до реализации

Что требуется оценивать

количество отказов при заполнении очередей
среднее время пребывания задачи в системе
среднее время отклика по приоритетам
загрузка процессоров

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.05.001 ПЗ

Лист

149

Цель и задачи проекта

Цель работы

Разработка и имитационное моделирование многопроцессорных приоритетных планировщиков задач с глобальной и локальными очередями.

- 1

проанализировать алгоритмы приоритетного планирования
- 2

формализовать работу планировщиков на языке НДА
- 3

разработать VHDL-модели шести вариантов планировщиков
- 4

провести моделирование в Xilinx 9.1 и сравнить характеристики

3

Разработанные модели планировщиков

В работе разработаны шесть VHDL-моделей, отличающихся структурой очередей и режимом обслуживания задач.

Модели с глобальной очередью

- планировщик с глобальной очередью FIFO;
- планировщик с глобальной очередью и относительными приоритетами;
- планировщик с глобальной очередью и абсолютными приоритетами.

Модели с локальными очередями

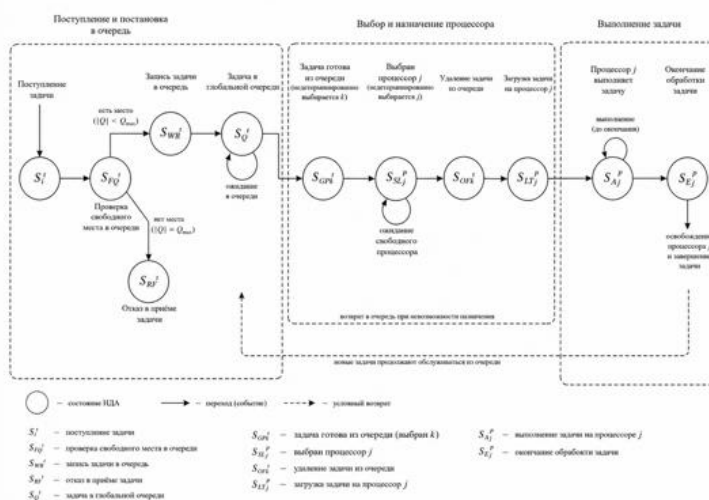
- планировщик с локальными очередями FIFO;
- планировщик с локальными очередями и относительными приоритетами;
- планировщик с локальными очередями и абсолютными приоритетами.

Все модели реализованы на языке VHDL и исследованы в среде Xilinx 9.1

4

Формализация алгоритмов на языке НДА

НДА модели планировщика с глобальной очередью



Роль формализации

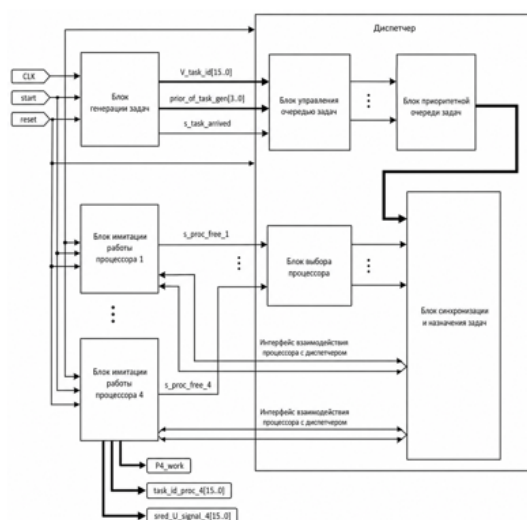
задаёт последовательность событий обслуживания
отражает ожидание в очереди и
назначение процессора
используется как основа для VHDL-
моделей

Ключевые этапы

поступление задачи
постановка в очередь
выбор задачи и процессора
выполнение и завершение

5

Модель планировщика с глобальной очередью



Планировщик с глобальной очередью задач



Принцип работы

все поступающие задачи помещаются в единую очередь
 блок выбора определяет подходящий процессор
 процессоры получают задачи из общей структуры

1

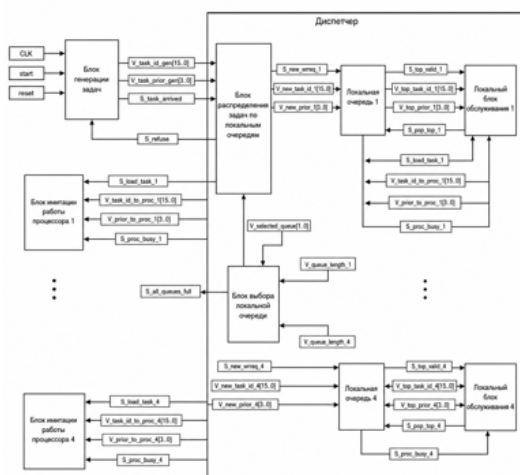
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.05.001 ПЗ

Лист

151

Модель планировщика с локальными очередями



Планировщик с локальной очередью задач



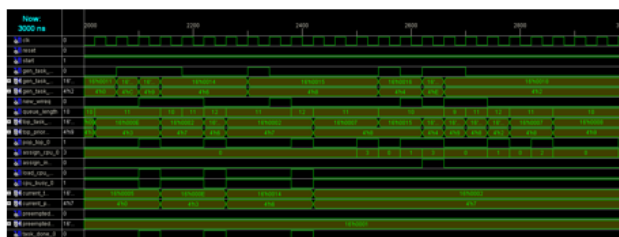
Отличие от глобальной очереди

задача сначала распределяется по одной из локальных очередей
каждая очередь связана со своим процессором
уменьшается конкуренция за общую структуру данных

7

Временные диаграммы работы моделей

Глобальная очередь



Локальные очереди



8

Условия имитационного моделирования

Интенсивность входного потока	$\lambda = 0,4; 0,3; 0,18; 0,125$
Среднее время обслуживания	$v = 9,5$ такта
Число процессоров	$n = 4$
Модельное время	3 000 000 тактов
Период синхросигнала	40 нс
Среда моделирования	Xilinx 9.1

Интервал между поступлениями задач
 $x = -1 / \lambda \cdot \ln(1 - s)$

Коэффициент загрузки системы
 $\rho = \lambda \cdot v / n$

s — равномерно распределённая случайная величина на интервале (0; 1)

9

Сводное сравнение результатов моделирования

Показаны общие характеристики для двух уровней загрузки системы

Модель	Отклик $\rho \approx 0,95$	Отказы $\rho \approx 0,95$	Отклик $\rho \approx 0,71$	Отказы $\rho \approx 0,71$
Гл. ABS	27.11	95313	8.93	4120
Гл. OTN	23.85	108905	7.22	3194
Гл. FIFO	23.5	110185	7.28	3590
Лок. ABS	26.35	114049	12.32	6722
Лок. OTN	24.1	82731	11.06	3785
Лок. FIFO	24.33	82975	11.18	3754

Основные выводы

- глобальная очередь даёт меньший средний отклик при средней загрузке;
- при высокой загрузке локальные очереди в OTN и FIFO дают меньше отказов;
- ABS снижает отклик высокоприоритетных задач, но ухудшает низкоприоритетные;
- FIFO прост, но не обеспечивает приоритетного обслуживания.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.05.001 ПЗ

Лист

153

Среднее время отклика по приоритетам

Загрузка $\rho \approx 0.95$: меньшее значение соответствует более быстрому началу обслуживания

Прим.	Гл. ABS	Гл. OTN	Гл. FIFO	Лок. ABS	Лок. OTN	Лок. FIFO
0	1	2.25	23.39	1.37	9.45	24.31
1	1	2.3	23.53	2.05	10.02	24.36
2	1.03	2.79	23.5	2.84	10.73	24.33
3	1.03	3.17	23.49	3.72	11.54	24.01
4	1.13	3.64	23.46	4.68	12.38	24.84
5	1.3	4.26	23.53	5.96	13.44	24.36
6	1.57	4.97	23.49	7.36	14.54	24.34
7	2.04	6.02	23.49	8.82	15.78	24.04
8	2.83	7.42	23.43	10.93	17.63	24.28
9	4.16	9.3	23.55	13.66	19.61	24.29
10	6.24	11.89	23.55	17.61	22.11	24.39
11	9.79	16.3	23.59	21.93	25.36	24.17
12	16.39	22.67	23.44	29.34	30.16	24.24
13	29.90	35.14	23.53	42.93	37.6	24.34
14	64.75	63.39	23.43	68.42	50.85	24.35
15	285.2	185.03	23.54	178.8	84.26	24.47

Что видно по таблице

- приоритет 0 обслуживается быстрее всего в ABS: 1,0 такта для глобальной очереди;
- у FIFO отклик почти одинаковый для всех приоритетов, так как приоритеты не учитываются;
- для приоритета 15 в ABS отклик резко увеличивается: низкоприоритетные задачи ожидают дольше;
- глобальная очередь лучше распределяет задачи между процессорами, локальные очереди зависят от выбранной очереди.

Отказы по приоритетам

Загрузка $\rho \approx 0.95$: показано количество задач, не принятых в очередь

Прим.	Гл. ABS	Гл. OTN	Гл. FIFO	Лок. ABS	Лок. OTN	Лок. FIFO
0	5931	6732	6925	7101	5194	5286
1	5906	6795	6864	7190	5233	5297
2	6058	6959	7015	7121	5116	5261
3	5973	6737	6857	7149	5243	5172
4	5995	6800	6990	7093	5231	5152
5	5969	6711	7020	7154	5103	5121
6	5990	6844	6913	6983	5060	5096
7	5949	6732	6775	7159	5219	5196
8	5870	6855	6941	7087	5267	5247
9	6003	6736	6784	7166	5114	5221
10	6040	6853	6899	7303	5087	5315
11	5986	6796	6686	7108	5083	5182
12	5958	6804	6736	7193	5214	5124
13	5889	6895	6876	6999	5261	5089
14	5824	6774	6874	7130	5086	5050
15	6012	6864	6946	7113	5254	5144

Вывод по отказам

- при высокой загрузке отказы появляются во всех моделях;
- наименьшее общее число отказов получено для локальных очередей в режиме относительных приоритетов и FIFO;
- отказы распределены по приоритетам близко равномерно, так как отказ связан с заполнением очереди;
- для ABS с локальными очередями число отказов выше из-за заполнения отдельных локальных очередей.

Вывод об эффективности структуры очередей и режимов обслуживания

Организация	Преимущества	Ограничения	Область применения
Глобальная очередь	меньший средний отклик при $r \approx 0,71$; равномерное получение задач процессорами	наше конкурентная за одну очередь; при высокой загрузке в OTN и FIFO отказов больше	когда важны балансировка и устойчивое среднее время отклика
Локальные очереди	меньше отказов при высокой загрузке в OTN и FIFO; меньше обременений общей структуре	возможна неравномерность заполнения очередей; для ABS больше отказов	когда важна простота доступа и снижение конкуренции

Режим	Влияние на обслуживание
FIFO	даёт простое и равномерное обслуживание, но не учитывает важность задач
Относительные приоритеты	улучшают полагание высокоприоритетных задач без прерывания выполняемой задачи
Абсолютные приоритеты	минимизируют отклик высокоприоритетных задач за счёт вытеснения, но увеличивают ожидание низкоприоритетных задач

Итог: универсально лучшей модели нет; выбор зависит от критерия – для быстрого обслуживания важных задач выгоден ABS, для меньшего числа отказов при высокой загрузке предпочтительны локальные очереди в режимах OTN и FIFO.

Итоги работы

Разработано: 6 VHDL-моделей планировщиков задач

Проверено: временные диаграммы работы глобальной и локальных очередей

Получено: значения времени отклика, времени пребывания, отказов и загрузки процессоров

Сравнено: влияние структуры очередей и режимов обслуживания

Вывод: выбор планировщика должен выполняться по требуемому критерию: минимальный отклик важных задач или минимизация отказов

Спасибо за внимание!

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.05.001 ПЗ

Лист

155