МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет вычислительной техники

Кафедра Вычислительная техника

Направление подготовки 09.04.01 «Информатика и вычислительная техника»

Магистерская программа «Вычислительные машины, комплексы, комплексы, системы и сети»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

на тему

Исследование характеристик подсистемы планирования/диспетчеризации задач на моделях массового обслуживания

Студент Данк 14.06.2018 (подпись, дата) Данилов Виталий Николаевич (ФИО полностью)

Руководитель 21.06.18 (подпись, дата) Бикташев Р. А. (фамилия, инициалы)

Нормоконтролёр 14.06.18 (подпись, дата) А.В. Кучин (фамилия, инициалы)

Рецензент 21.06.18 (подпись, дата) П.С.Чернов (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 25.05.2018г. № 38-09-12)

Заведующий кафедрой (подпись) Д.В. Пащенко (фамилия, инициалы)

Работа защищена с отметкой хор. (протокол заседания ГЭК от 15.06.18 № 17)

Секретарь ГЭК (подпись) Е.И. Гурин (фамилия, инициалы)

Пенза, 2018

Реферат

Пояснительная записка 116 листов, 3 таблицы, 31 рисунок.

Ключевые слова: ОЧЕРЕДЬ, ПЛАНИРОВЩИК, ОПЕРАЦИОННАЯ СИСТЕМА, ДИСПЕТЧЕР, МНОГОПРОЦЕССОРНАЯ СИСТЕМА, ПРОЦЕССОР, АЛГОРИТМ ПЛАНИРОВАНИЯ, МЕТОД ДИСПЕТЧЕРИЗАЦИИ.

Данная магистерская диссертация посвящена разработке программы моделирования стохастических сетей массового обслуживания, моделированию алгоритмов диспетчеризации много процессорных планировщиков задач.

В работе была разработана программа на языке C#,проведено моделирование алгоритмов бесприоритетной диспетчеризации с ограничением длины очереди и приоритетной диспетчеризации без ограничения длины очереди.

Промоделированы и проанализированы алгоритмы диспетчеризации. Произведено сравнение промоделированных алгоритмов.

Содержание

Введение 4
1. Формализация и моделирование алгоритмов управления процессами в многопроцессорных системах 9
1.1. Формализация классической задачи «о работе парикмахерской» на основе НДА....9
1.2. Проблема спящего парикмахера и задача о работе парикмахерской....9
Задача о работе парикмахерской....11
1.3. Формализация алгоритма управления процессами в задаче «о спящем парикмахере» на основе НДА. 19
1.4. Формализация алгоритма управления процессами в задаче «работа парикмахерской» на основе НДА. 23
1.5. Структура аппаратного планировщика/ диспетчера с глобальной очередью задач....29
Выводы к первой главе. 42
2. Аналитическое моделирование планировщика/диспетчера задач 43
2.1 Анализ методов планирования и диспетчеризации задач современных операционных систем 43
2.2 Аналитическое моделирование диспетчеров задач с однородным входящим потоком задач, бесприоритетной очередью с ограничением числа мест 52
2.3 Аналитическое моделирование планировщиков/диспетчеров задач с неоднородным потоком задач 57
2.4 Расчет характеристик планировщиков/диспетчеров задач с бесприоритетным обслуживанием и ограничением числа мест в очередях 64
3. Разработка программы расчета характеристик стохастических сетей массового обслуживания 73
3.1. Алгоритм расчета характеристик систем и сетей массового обслуживания 73
3.2. Алгоритм расчета характеристик многоканальных СМО с приоритетным обслуживанием....78
3.3. Структура входных и выходных данных....81
3.4 Разработка алгоритмов решения задачи 82
3.5 Описание программы....87
Заключение....91
Список литературы 92
ПриложениеА....95

Введение

Для функционирования передовых вычислительных систем в режиме реального времени, практически регулярно применяют многопроцессорные, а помимо этого и многоядерные архитектуры, которые создаются вместе с решением задач управления процессами и ресурсами с использованием диспетчеризации. Эффективность работы функций диспетчеров задач прибывает в зависимости от скорости переключения процессов. Частота вызова приложениями планировщика и диспетчера, в многопроцессорной системе существенно выше, нежели в однопроцессорных системах, и именно по данной причине обращение к общим ресурсам формируют конфликтные ситуации. Значительно чаще подобное явление представляется возможным в многоядерных системах на кристалле, которые обладают потенциально высокой производительностью. Традиционная программная реализация диспетчеров задач далеко не всегда предоставляет требуемую производительность системы. В соответствии с данной причиной, в процессе проектирования ядра операционной системы требуется выбирать более эффективные технические средства его разработки.

Задача диспетчеризации процессов в многопроцессорной системе выступает в качестве двумерной. В период работы функции назначения задачи осуществляется функция выделения целевого процессора, вследствие чего возникают дополнительные издержки времени. Помимо этого, в диспетчере задач с разделением во времени формируется процесс перезагрузки кэш-памяти, который взаимосвязан с переключением задач, когда прерванная задача с достаточно высокой вероятностью способна быть направлена для продолжения обслуживания в иной процессорный узел. Данный процесс повышает частоту кэш-промахов, и в абсолютно любом случае отрицательно воздействует на производительность многопроцессорной системы. Другое противоречие в диспетчерах задач с разделением в пространстве, которое требуется разрешить, взаимосвязано с тем, что автоматическое поддержание различного уровня нагрузки процессорных узлов невозможно. Решение представленной задачи лежит в плоскости принудительного обеспечения баланса методами планировщика/диспетчера, что в любом случае

порождает дополнительные временные расходы на исполнение определенной программы балансирования. Представленные выше факторы неизбежно формируют рост трудности программной реализации ядра операционных систем, приводят к уменьшению общей производительности многопроцессорной системы, а помимо этого уменьшают надежность и безопасность.

Во время обращения к критическим ресурсам операционной системы, а помимо этого и в процессе планирования и диспетчеризации задач к данным ресурсам возникают очереди процессов, которые ожидают выполнения иных процессов. Согласно данной причине для аналитического описания многопроцессорной системы, зачастую, используется теория массового обслуживания. В подобном случае математическая модель многопроцессорной системы представляется в качестве стохастической сети массового обслуживания, в которой роли узлов берут на себя модели ресурсов, а заявки – запросы на выполнение пользовательских программ.

Преимущественно большая часть задач проектирования многопроцессорных вычислительных систем взаимосвязаны с распределением и диспетчеризацией ресурсов среди конкурирующих требований, притом, что моменты возникновения данных требований носят случайный характер. Правильным решением данных задач выступает математическое моделирование многопроцессорных систем, которые включают в себя диспетчеры задач, что предоставит возможность ускорить процесс проектирования диспетчеров для специальных многопроцессорных систем, а помимо этого уменьшить себестоимость проекта. Представленные аспекты обуславливают актуальность выбранной темы исследования.

Объект исследования – планировщики и диспетчеры задач в многопроцессорных вычислительных системах.

Предмет исследования – математическое и функциональное моделирование планировщиков/диспетчеров вычислительных процессов с использованием языка недетерминированных автоматов (НДА), разомкнутых систем и сетей массового обслуживания и на языке моделирования аппаратуры VHDL.

Цель данной работы – получение временных характеристик и сравнительная оценка планировщиков/диспетчеров задач с программной и аппаратной реализацией.

С целью достижения поставленной цели в диссертационной работе решаются следующие задачи:

1) разработка автоматной модели на основе языка НДА и на её основе VHDL-модели планировщика/диспетчера задач и оценка характеристик аппаратной реализации;

2) разработка аналитической модели планировщика/диспетчера задач для получения его вероятностно-временных характеристик;

3) разработка программы для расчета характеристик разомкнутых сетей массового обслуживания;

4) проведение вычислительных экспериментов с использованием разработанных моделей.

Методы исследований. Для того чтобы решить поставленные задачи в диссертационной работе применены методы:

- теории аналитического и имитационного моделирования;
- теории вероятностей;
- теории систем и сетей массового обслуживания.

Научная новизна работы состоит в следующем:

1) разработана на основе языка НДА автоматная модель распределенного планировщика/диспетчера задач с разделением времени в составе многопроцессорной системы. На основе НДА- модели создана VHDL – программа аппаратной реализации указанного планировщика/диспетчера задачи произведена оценка его характеристик;

2) разработана математическая модель распределенного планировщика/диспетчера задач с разделением во времени в составе многопроцессорной системы, которая принимает во внимание задержки, взаимосвязанные с конфликтами за доступ к очереди задач, переключением

контекстов, а также ограничение числа мест в очередях обслуживающих устройств;

3) разработана программа для расчета вероятностно-временных характеристик подсистемы «диспетчер задач – процессорный узел».

Практическая значимость работы.

Применение предложенных аналитической и программной моделей обеспечивает сравнительную оценку временных характеристик планировщика/диспетчера задач на этапе эскизного проектирования, без проведения дорогостоящих физических экспериментов.

Реализация и внедрение результатов работы. Методы моделирования диспетчеров задач многопроцессорных систем будут применены в учебном процессе кафедры «Вычислительная техника» Пензенского государственного университета при осуществлении основной образовательной программы по дисциплинам «Вычислительные системы» для студентов, которые обучаются по направлению «Информатика и вычислительная техника».

Достоверность результатов работы подтверждается имитационным моделированием, применением популярных методов математического моделирования, а помимо этого докладами на научных конференциях и публикацией в научных журналах.

На защиту выносятся.

1. Методы математического моделирования диспетчеров задач с разделением во времени, а кроме того и разделением в пространстве в составе многопроцессорных систем, которые принимают во внимание потери времени, взаимосвязанные с конфликтами, ограниченностью ресурсов, балансировкой загрузки процессорных узлов и приоритетностью очередей.

2. Улучшенный численный метод и вычислительный алгоритм оценки производительности многопроцессорной системы с диспетчером задач со стратегией разделения времени, которые учитывают приоритетность входящего потока задач.

3. Комплекс проблемно-ориентированных программ для расчета характеристик диспетчеров задач в многопроцессорных вычислительных системах, которые основаны на разомкнутых сетях массового обслуживания.

4. Результаты комплексных исследований вероятностно-временных характеристик диспетчеров задач с использованием разработанных математических методов и комплекса программ.

1. Формализация и моделирование алгоритмов управления процессами в многопроцессорных системах

1.1. Формализация классической задачи «о работе парикмахерской» на основе НДА

В процессе работы компьютера в многозадачном режиме, порой, запускается одновременно несколько процессов, либо же потоков, которые претендуют на применение центрального процессора. Такое осуществляется в том случае, когда в состоянии готовности сразу прибывают два, либо больше процесса, либо потока. В случае, когда доступен исключительно один центральный процессор требуется выбрать, какой из данных процессов будет исполняться следующим. Непосредственно, та часть операционной системы, на которую возлагается данный выбор, именуется планировщиком, а что касаемо алгоритма, который ею используется, именуется, непосредственно, алгоритмом планирования.

Большинство однотипных вопросов, которые применяются к планированию процессов, способны использоваться еще и к планированию потоков, не смотря на то, что имеются и некоторые отличия. В случае, когда ядро управляет потоками, планирование, как правило, касается абсолютно каждого из потоков, практически не различая, к какому, непосредственно, процессу они принадлежат (либо все же делая незначительные различия). Мы рассмотрим традиционные проблемы межпроцессорного взаимодействия.

1.2. Проблема спящего парикмахера и задача о работе парикмахерской

В парикмахерской существует один брадобрей, кроме того его кресло и «n» количество стульев для клиентов. В случае если желающих воспользоваться его услугами не существует, то парикмахер располагается в собственном кресле и спит. Если в парикмахерскую входит посетитель, он обязан разбудить парикмахера. В случае если посетитель приходит и наблюдает, что парикмахер занят, то он, либо садиться на стул (при наличии места), или же покидает заведение (при отсутствии места). Требуется запрограммировать парикмахера и посетителя, таким образом, чтобы избежать состояния состязания. У представленной задачи имеется множество аналогов в области массового

обслуживания, к примеру: информационная служба, которая обрабатывает в один промежуток времени ограниченное количество запросов, с компьютеризированной системой ожидания для запросов.

Рисунок 1. Спящий парикмахер.

В представленном решении применяется три семафора, к которым относятся следующие: «customers», с целью подсчета ожидающих клиентов (посетитель, который сидит в кресле парикмахера не принимается в учет, поскольку он уже не ждет); «barbers», число парикмахеров (0, либо 1), которые простаивают в ожидании посетителя; «mutex» с целью осуществления взаимного исключения. Помимо этого применяется переменная waiting, которая предназначена с целью подсчета ожидающих клиентов. Она считается копией переменной customers. Наличие в программе данной переменной взаимосвязано с тем фактом, что прочитать текущее значение семафора не представляется возможным. В представленном решении клиент, который заглядывает в парикмахерскую, обязан сосчитать число очередь ожидающих клиентов. В случае, если клиентов меньше чем стульев, новый клиент остается, а в ином случае – покидает парикмахерскую.

Когда парикмахер приходит на работу, он осуществляет процедуру barber, тем самым блокируясь на семафоре customers, так как значение семафора равняется 0. После парикмахер засыпает и спит до того момента, пока не зайдет первый посетитель.

Приходя в парикмахерскую, клиент исполняет процедуру customer, запрашивая тем самым доступ к mutex для входа в критическую сферу. В случае если следом за ним появится еще один клиент, ему не получится что-то сделать до того момента, пока первый клиент не освободит доступ к mutex. После клиент осуществляет проверку наличия свободных стульев, а в случае неудачи, освобождает доступ к mutex и покидает парикмахерскую.

В случае если имеется свободный стул, то клиент повышает значение целочисленной переменной waiting. После он исполняет процедуру up на семафоре customers, тем самым активируя поток посетителей к парикмахеру. В данный момент оба – посетитель и парикмахер – являются активны. В случае, когда клиент освобождает доступ к mutex, парикмахер захватывает его, а помимо этого проделывает определенные служебные операции, и далее приступает к стрижке посетителя.

По завершении стрижки клиент выходит из процедуры и покидает помещение. В отличие от ранее представленных программ, цикла клиентов не существует, так как каждого клиента стригут исключительно один раз. Цикл парикмахера имеется, и парикмахер пытается отыскать следующего клиента. В случае если ему это представляется возможным, то он стрижет следующего клиента, а в ином случае – парикмахер засыпает.

Требуется отметить, что, невзирая на отсутствие передачи сведений в проблеме читателей и писателей, а помимо этого и в проблеме спящего парикмахера, обе указные проблемы относятся к проблемам межпроцессорного взаимодействия. Так как требуют синхронизации нескольких процессов.

Задача о работе парикмахерской

В парикмахерской располагается три кресла, три парикмахера, а помимо этого и зал ожидания, в котором четыре посетителя способны расположиться на

диване, а другие – стоя. Существует ограничение на общее число посетителей внутри помещения.

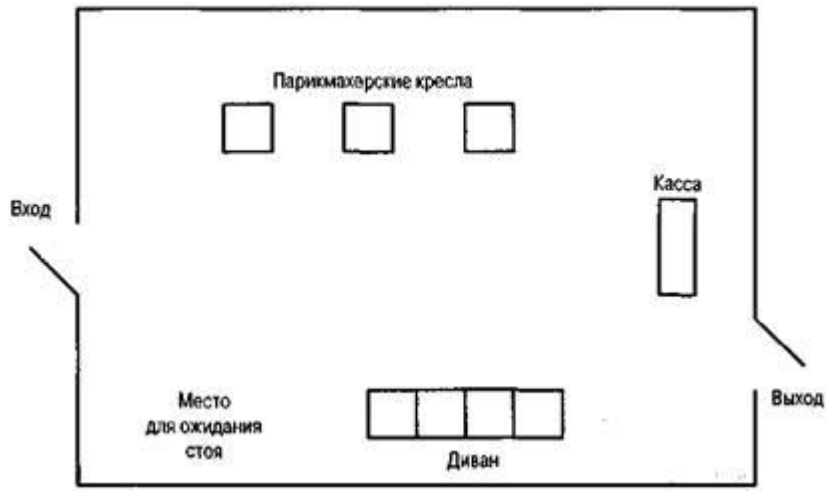


Рисунок 2. Парикмахерская.

Посетитель не способен войти в парикмахерскую, в случае если она заполнена полностью иными посетителями. Оказавшись внутри, посетитель или присаживается на диван, или стоит, в случае если последний занят. В момент, когда парикмахер освобождается, к нему отправляется более продолжительно ожидающий посетитель с дивана; в случае если существуют стоящие посетители, то тот из них, кто ожидает дольше иных, располагается на диване. По завершении стрижки приять оплату способен абсолютно любой парикмахер, однако поскольку кассир в парикмахерской только один, то оплата принимается в один момент времени исключительно от одного посетителя. Рабочее время парикмахера делится на стрижку, принятие оплаты от посетителя, а помимо этого и сон в собственном кресле.

Неполное решение задачи о парикмахерской

На примере ниже продемонстрирована реализация парикмахерской с применением семафоров; подразумевается, что абсолютно все очереди семафоров обрабатываются согласно принципу «первым вошел - первым вышел». В листинге с целью экономии места функции располагаются в два столбца.

```
Semaphoremax_capacity = 20;
semaphoresofa = 4;
semaphorebarber_chair = 3;
semaphorecoord = 3;
semaphorecust_ready = 0,   finished =0,
leave_b_chair = 0,         payment = 0,
receipt = 0;
```

```
voidcustomer()
{
wait(max_capacity);
enter_shop ();
wait(sofa);
sit_on_sofa();
wait(barber_chair);
get_up_from_sofa ();
signal(sofa);
sit_in_barber_chair();
signal(cut_ready);
wait(finished);
leave_barber_chair () ;
signal(leave_b_chair);
pay();
signal(payment);
wait(receipt);
exit_shop();
signal(max_capacity);
}
```

```
voidbarber()
{
while(true)
{
wait(cust_ready);
wait(coord);
cut_hair();
signal(coord);
signal(finished) ;
wait(leave_b_chair);
signal(barber_chair);
}
}
voidcashier ()
{
while(true)
{
wait(payment);
wait(coord);
accept_pay();signal(coord);
signal(receipt);
}
}
```

```
voidmain ()
{
parbegin(customer,[50  раз], customer,barber,barber,barber,cashier);
}
```

Рисунок 3. Пример неполного решения задачи о парикмахерской.

Базовое тело программы активизирует процессы порядка 50-ти посетителей, трех парикмахеров и одного кассира. Далее рассмотрим назначение всевозможных синхронизирующих операторов.

Вместимость парикмахерской и дивана. Вместимость парикмахерской и дивана управляется следующими семафорами, такими как: max_capacity и sofa. Абсолютно каждый раз при попытке посетителя войти в парикмахерскую значение семафора max_capacity сокращается на 1; когда посетитель покидает парикмахерскую, оно возрастает. В случае если парикмахерская заполнена, то процесс посетителя приостанавливается функцией wait (max_capacity). Таким же образом обрабатывается еще и попытка сесть на диван.

Емкость парикмахерских кресел. В наличии существует три парикмахерских кресла, и необходимо обеспечить их корректное применение. Семафор barber_chair предоставляет гарантию одновременного обслуживания не свыше трех посетителей таким образом, чтобы один посетитель не оказался на коленях у иного. Посетитель не встанет с дивана до тех пор, пока не окажется свободным хоть одно кресло (вызов wait (barber_chair)), а абсолютно каждый парикмахер уведомляет том, что его кресло освободилось (вызов signal (barber_chair)). Корректный доступ к парикмахерским креслам гарантируется организацией очередей семафоров, который первым был блокирован в очереди, и, соответственно, первым же приглашается пройти на стрижку. Требуется отметить, что если в процедуре посетителя вызов wait (barber_chair) расположить после signal (sofa), то абсолютно каждый посетитель будет исключительно присаживаться на диван, после чего незамедлительно вскакивать и занимать первоначальную позицию у кресла, формируя тем самым, лишнюю толкотню и мешая трудиться парикмахеру.

Размещение посетителя в кресле. Семафор cust_ready предоставляет подъем спящего парикмахера, уведомляя его о новом посетителе. Без данного семафора парикмахер никогда бы не отдыхал и начинал стрижку незамедлительно, после того, как очередной посетитель встанет с кресла. Следовательно, если клиент отсутствует, то в данный момент, парикмахер осуществлял бы стрижку воздуха.

Удержание посетителя в кресле. В случае если уж посетитель оказался в кресле, то он обязан отсидеть там до завершения стрижки, о чем, непосредственно, просигнализирует семафор finished.

Ограничение числа посетителей в креслах. Семафор barber_chair предназначается с целью ограничения числа посетителей в креслах – их не должно быть свыше трех. Но одного семафора barber_chair для этого является недостаточно. Посетитель, который не приобретет процессорное время, сразу после того, как парикмахер сообщит об окончании работы над его прической (вызов signal (finished)), останется в кресле (к примеру, впав в транс, либо же глубоко задумавшись о задаче, о парикмахерской), в то время как данном же кресле будет стараться расположиться новый посетитель. С целью решения представленной задачи применяется семафор leave_b_chair, который не предоставляет возможность парикмахеру пригласить нового посетителя до того момента, пока предыдущий не освободит кресло.

Оплата стрижки. Безусловно, с денежными средствами необходимо быть предельно осторожным. Кассир обязан быть уверен в том, что абсолютно каждый посетитель, покидая парикмахерскую, в первую очередь, расплатиться, а каждый посетитель, совершив оплату стрижки, обязан получить чек. Это достигается путем передачи денежных средств кассиру из рук в руки – каждый посетитель, покинув кресло, совершает оплату услуг парикмахера, после чего уведомляет об этом кассира (вызов signal (payment)) и ожидает приобретения кассового чека (вызов wait (receipt)). Кассир реализует прием платежей, дожидается сигнала о платеже, принимая денежные средства, а после сообщая об этом. Здесь требуется постараться избежать перечня программных ошибок. В случае если вызов signal (payment) исполняется перед вызовом pay (), то посетитель способен оказаться прерванным в данный момент, и кассир, в свою очередь, будет пытаться принять не переданные денежные средства. Помимо этого существует еще наиболее существенная ошибка, которая происходит в процессе обмена строк signal (payment) и wait (receipt). Это способно привести к взаимоблокировке абсолютно всех посетителей и кассира их операциями wait. Регулирование действий кассира и

парикмахера. Данное действие осуществляет парикмахер, когда не реализует стрижку посетителя. С целью того, чтобы обеспечить исполнение парикмахером в один период времени исключительно одной функции, применяется семафор coord. Назначение абсолютно каждого семафора программы представлено в таблице 1.1.

Таблица 1. Назначение семафоров.

Семафор	Операция wait	Операция signal
max_capacity	Посетитель ждет возможности войти в парикмахерскую	Посетитель, который покидает парикмахерскую, уведомляет об этом ожидающего входа
sofa	Посетитель ждет возможности присесть на диван	Посетитель, который встает с дивана, уведомляет об этом ожидающего возможность присесть на диван
barber_chair	Посетитель ждет пустое кресло	Парикмахер уведомляет, что кресло свободно
Cust_read	Парикмахер ждет, пока посетитель сядет в кресло	Посетитель уведомляет парикмахера, что он уже сел в кресло
finished	Посетитель ждет завершения стрижки	Парикмахер уведомляет посетителя, что стрижка завершена
Leave_b_chair	Парикмахер ждет, пока посетитель освободит кресло	Посетитель уведомляет парикмахера о том, что он встал с кресла
payment	Кассир ждет оплаты услуг посетителем	Посетитель уведомляет парикмахера о том, что он совершил оплату стрижки
receipt	Посетитель ждет кассовый чек	Кассир уведомляет о том, что оплата принята
coord	Ожидание освобождения парикмахера для стрижки или для исполнения обязанностей кассира	Сигнал об освобождении парикмахера

Процесс кассира можно ликвидировать, внедряя функцию оплаты в процедуру парикмахера. Абсолютно каждый парикмахер будет поэтапно

производить стрижку и принимать оплату за нее. Но при имении одного кассового аппарата требуется ограничить доступ к функции accept_pay() только одним парикмахером в абсолютно каждый период времени. Этого представляется возможным добиться, рассматривая функцию, непосредственно, как критический раздел, а помимо этого оградив ее определенным семафором.

Полное решение задачи о парикмахерской

Невзирая на приложенные усилия у нас остались некоторые трудности. Решение одной из проблем заложено в оставшейся части подраздела.

В неполноценном решении прибывает проблема, которая способна привести к некорректной работе с посетителями. Представим, что на данный момент в парикмахерских креслах расположены три посетителя. Они, скорее всего, заблокированы вызовами wait (finished), и согласно организации очереди будут деблокированы согласно тому порядку, в котором они садились в кресла. Но что, если один из парикмахеров очень оперативно трудится (либо же один из посетителей лысый)? Следовательно, произойдет ситуация, когда одного посетителя будут вытаскивать из кресла и заставлять совершить оплату за незавершенную прическу, в то время как иного, полноценно постриженного посетителя буду держать в кресле силой.

Представленная проблема разрешается путем добавления новых семафоров. Мы присваиваем абсолютно каждому пользователю номер (ровно как, если бы при входе в парикмахерскую, абсолютно каждый посетитель получал бы номерок). Семафор mutexl предоставляет защиту доступа к глобальной переменной count, гарантируя тем самым уникальность номера абсолютно каждого посетителя. Семафор finished преобразился в массив из 50ти семафоров. В случае когда посетитель присаживается в кресло, он исполняет инструкцию wait (finished[custnr]), ожидая свой семафор. По завершении стрижки парикмахер осуществляет инструкцию signal (finished[b_cust]), отпуская из кресла того посетителя, которого он стриг. Нам лишь остается исследовать, каким образом парикмахер узнает номер посетителя. Посетитель помещает собственный номер в очередь enqueue 1, непременно перед тем, как уведомить парикмахера о готовности посредством

семафора cust_ready. Когда парикмахер готов приступить к стрижке посетителя, dequeuel (b_cust) удаляет его номер посетителя из очереди, а помимо этого помещает его в локальную переменную парикмахера b_cust.

1.3. Формализация алгоритма управления процессами в задаче «о спящем парикмахере» на основе НДА.

Исследуя клиентов и парикмахера, непосредственно, как взаимодействующие процессы, для которых клиент представляет собой посетителя, который запрашивает сервиса у парикмахера. А парикмахер, в свою очередь, представляется в качестве сервера, который обеспечивает этот сервис. Указанный тип взаимодействия можно представить, непосредственно, как пример взаимоотношений посетитель-сервер.

Действия парикмахера и клиента требуется синхронизировать так, чтобы предоставить парикмахеру и клиенту встречу-рандеву, когда парикмахер обязан дождаться прихода клиента, а клиент, в свою очередь, освобождения парикмахера. При всем при этом, клиенту требуется ждать, пока парикмахер завершит его стрижку, в результате чего он освобождает кресло парикмахера и ждет открытия двери парикмахером. Непосредственно, в собственную очередь, парикмахер, перед тем как закрыть дверь, обязан подождать пока посетитель уйдет. Следовательно, парикмахер и клиент проходят черед последовательность синхронизированных стадий, которые начинаются с рандеву.

С целью формального описания алгоритма взаимодействия процессов в представленной задаче будем применять язык систем канонических уравнений. Для указанной цели введем основные частные события, которые осуществляются в представленном алгоритме, к которым относятся следующие:

$S_з^k$ - событие, определяющее приход нового клиента в парикмахерскую;

$S_{оч}^k$ - событие, определяющее наличие места в очереди к парикмахеру;

S_0^k и S_0^n - события, определяющие нахождение посетителя и парикмахера в салоне парикмахерской;

$S^{k}_{\text{БП}}$ и $S^{n}_{\text{БК}}$ - события, свидетельствующие о том, что посетитель будит парикмахера, а парикмахер будит посетителя;

$S^{k}_{\text{КП}}$ и $S^{n}_{\text{ОК}}$ - события, свидетельствующие о том, что посетитель присел в кресло парикмахера, а парикмахер освободил собственное кресло;

$S^{k}_{\text{С}}$ и $S^{n}_{\text{С}}$ - события, свидетельствующие о том, что посетитель и парикмахер спят;

$S^{k}_{\text{ГТ}}$ и $S^{n}_{\text{ГТ}}$ - события, свидетельствующие о том, что посетитель и парикмахер готовы к обслуживанию;

$S_{\text{Н}}$ и $S_{\text{К}}$ - события, свидетельствующие о начале и завершении стрижки;

$S^{k}_{\text{ОК}}$ и $S^{n}_{\text{ОД}}$ - события, свидетельствующие о том, что посетитель освободил кресло парикмахера, а парикмахер открыл выходную дверь;

$S^{k}_{\text{У}}$ и $S^{n}_{\text{ЗД}}$ - события, свидетельствующие о том, что посетитель покинул парикмахерскую, а парикмахер закрыл выходную дверь.

Основываясь на словесно представленном алгоритме управления процессами в задаче о спящем парикмахере и внедренных событиях, которые осуществляются в данном алгоритме, система канонических уравнений, характеризующих указанные события и соответствующий им град недетерминированного автомата (НДА) будут обладать таким видом:

- для процесса посетитель:

$$S^{k}_{0} = S^{k}_{\text{З}} S^{k}_{\text{ОЧ}},$$
$$S^{k}_{\text{С}}(t+1) = S^{k}_{0}\ \overline{S^{n}_{\text{С}}} \vee S^{k}_{\text{С}} \overline{S^{n}_{\text{БК}}},$$
$$S^{k}_{\text{БП}}(t+1) = S^{k}_{0}\ S^{n}_{\text{С}},$$
$$S^{k}_{\text{КП}}(t+1) = S^{k}_{\text{С}}\ S^{n}_{\text{БК}} \vee S^{k}_{\text{БП}} S^{n}_{\text{ОК}},$$
$$S^{k}_{\text{ГТ}}(t+1) = S^{k}_{\text{КП}}\ S^{n}_{\text{ЗД}} \vee S^{k}_{\text{ГТ}}\ \overline{S^{n}_{\text{ГТ}}};$$

- для процесса – сервер (парикмахер):

$$S^{n}_{С}(t+1) = S^{n}_{0}\overline{S^{k}_{0}} \vee S^{n}_{С}\,\overline{S^{k}_{БП}};$$
$$S^{n}_{БК}(t+1) = S^{n}_{0} S^{k}_{0} S^{k}_{С},$$
$$S^{n}_{ОК}(t+1) = (S^{n}_{0} S^{k}_{0}\overline{S^{k}_{С}} \vee S^{n}_{С}) S^{k}_{БП},$$
$$S^{n}_{ГТ}(t+1) = (S^{n}_{ОК} \vee S^{n}_{БК}) S^{k}_{КП} \vee S^{n}_{ГТ}\,\overline{S^{k}_{ГТ}}.$$

Система канонических уравнений, которая описывает события после рандеву:

$$S_{Н}(t+1) = S^{k}_{ГТ}\, S^{n}_{ГТ},$$
$$S_{К}(t+1) = S_{Н},$$
$$S^{k}_{ОК}(t+1) = S_{К},$$

$$S^{n}_{ОД}(t+1) = S_{К},$$
$$S^{k}_{У}(t+1) = S^{k}_{О}\, S^{n}_{ОД},$$
$$S^{n}_{ЗД}(t+1) = S^{n}_{ОД}\, S^{k}_{У}.$$

В представленных ранее системах канонических уравнений, а помимо этого и графе НДА (см.рис.1) были опущены события, которые соответствуют пустым событиям обратной связи в циклах из логических условий. Данная взаимосвязь символизирует ожидание определенного события S_i истинности логического условия, в процессе которого осуществляется переход к событию S_j, считающемся первым приемщиком события S_i. К примеру, рассмотрим переход от события $S^{k}_{ОК}$ к событию $S^{k}_{У}$, для которого выход условной вершины $S^{n}_{ОД}$ соединен, непосредственно, с ее входом.

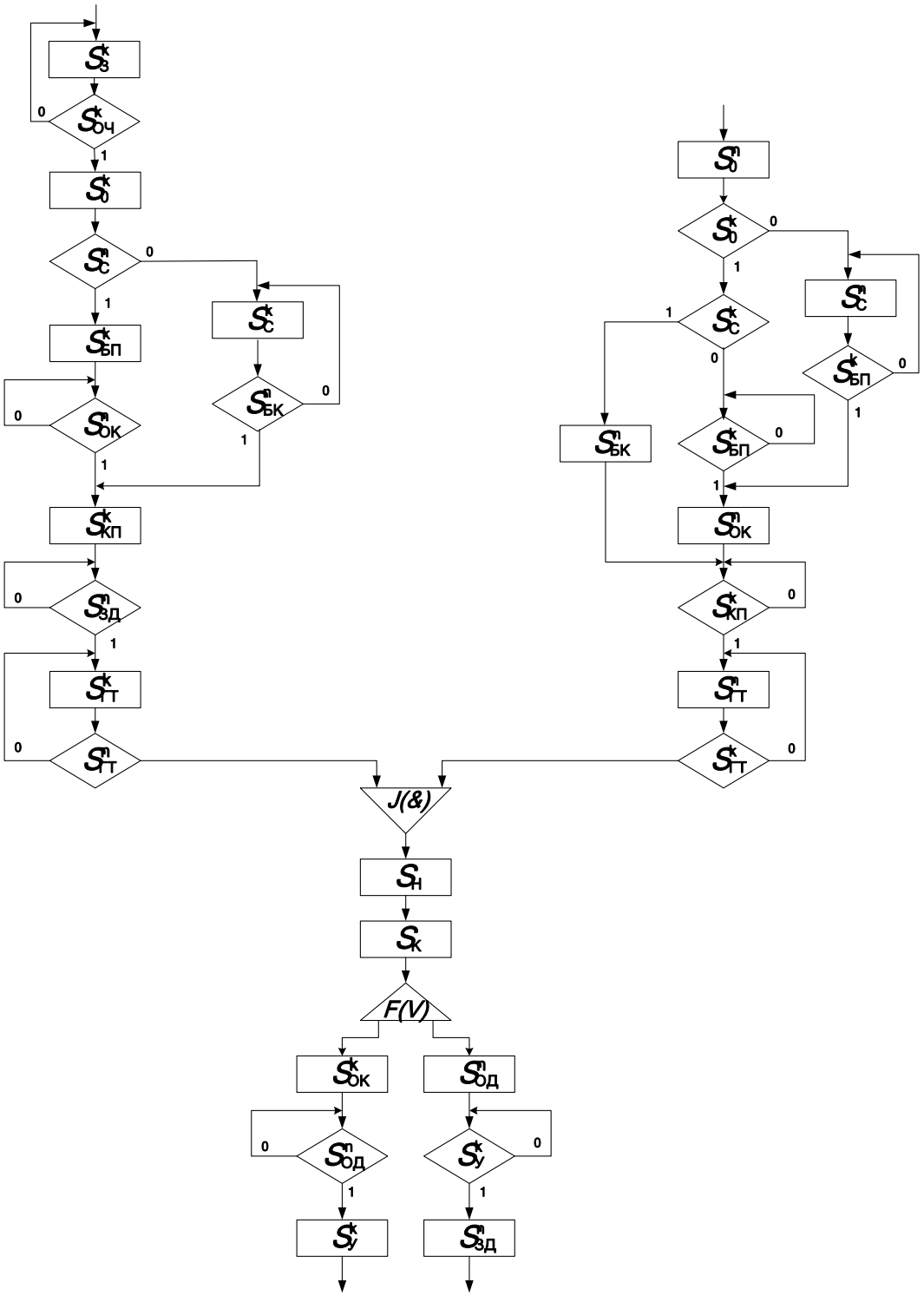


Рис.4. Граф НДА алгоритма управления взаимодействующими процессами в задаче о парикмахерской

Введя пустую операторную вершину в обратную связь, а помимо этого, обозначив её символом S_e, получим такие уравнения, как:

$$S_{\text{е}}(t+1) = \left(S_{\text{ок}}^{\text{k}} \vee S_{\text{е}}\right) S_{\text{од}}^{\text{n}},$$
$$S_{\text{у}}^{\text{k}}(t+1) = \left(S_{\text{ок}}^{\text{k}} \vee S_{e}\right) S_{\text{од}}^{\text{n}}.$$

Такие же уравнения способны быть представлены и для иных переходом, для которых выход условной вершины объединен с ее входом.

1.4. Формализация алгоритма управления процессами в задаче «работа парикмахерской» на основе НДА.

Традиционный подход, который взаимосвязан с таким обслуживанием является, аналогичен задаче «спящего парикмахера», который обширно использовался для вычислительных систем с одним процессором. Непосредственно, для многопроцессорных систем подобный подход обязан быть увеличен до задачи, которую представляется возможным именовать «работа парикмахерской». Представленный процесс иллюстрирует взаимоотношения «посетитель-сервер», которые облают местом между процессами в многопроцессорных вычислительных системах. При всем при этом аналогом процессоров выступает коллектив парикмахеров, а в качестве аналога задач выступают посетители парикмахерской. Алгоритм управления процессами в указанной задаче именуют алгоритмами планирования и диспетчеризации. Причем функция планирования состоит в постановке задач в очередь, согласно принятой дисциплиной обслуживания (FIFO, LIFO, приоритетная), а функция диспетчеризации – выборе обслуживающего процессора, а помимо этого и запуск очередной задачи из очереди в определенном выбранном процессоре.

В непосредственно, самом общем виде алгоритм планирования/диспетчеризации задач представляется таким образом: в первоначальном состоянии абсолютно каждый включающийся в деятельность процессор осуществляет операцию «готов» и переходит в спящий режим, в котором прибывает до того момента, пока на обслуживание не поступит задача из очереди. В случае если абсолютно все процессоры заняты обслуживанием, то снова поступившая задача помещается в самый конец очереди, количество мест в которой является ограниченным. В случае если снова поступившая задача выявит,

что очередь является заполненной, она на обслуживание не принимается и как следствие, покидает систему. Что касаемо принятой на обслуживание задачи, то она прибывает в очереди до того момента, пока не поступит на исполнение в процессор, при всем при этом, в очереди, освобождается одно место. По завершении выполнения очередной задачи планировщик/диспетчер изучает очередь, и в случае если в ней существуют ожидающие, то он назначает на осуществление в свободный процессор ту, которая располагается в голове списка (согласно принципу *FIFO*). Если очередь является пустой, то свободные процессоры перенаправляются в режим ожидания. Если в процессорном пуле существует несколько свободных процессоров, то осуществляется приоритетный выбор одного из них с целью облуживания очередной задачи.

В общем случае алгоритм планирования/диспетчеризации в явном виде взаимосвязан с взаимодействием процессов. С одной стороны задачи (например, процессы, потоки), которые требуют собственного выполнения, а с иной стороны – обслуживающие их процессорные узлы. Представленные действия требуется синхронизировать так, чтобы обеспечить так именуемое «рандеву», когда *j*–й процессорный узел обязан дождаться поступления задачи, а *i*-я процесс, в свою очередь – освобождения одного из процессоров, вследствие чего она будет обслуживаться на протяжении определенного времени. Под «рандеву» подразумевается условная синхронизация двух процессов, когда условием продолжения их осуществления выступает одновременная готовность обоих. Рандеву осуществляется конъюнктивным соединением J(&).

Формализация алгоритмов управления параллельными процессами осуществлена на базе применения механизма «рандеву». Абсолютно весь алгоритм деятельности диспетчера содержит, непосредственно, три части: клиентская (постановка задачи в очередь), северная (обслуживание процессорами), а помимо этого и «рандеву» (наличие задачи в очереди, а кроме того и готовность одного из процессоров к обслуживанию данной задачи). С целью описания алгоритма введены основные части события, которые проиллюстрированы в таблице 1.4. В

последней колонке приведено их соответствие согласно сигналам на схеме устройства диспетчеризации задач.

Таблица 2. Отмеченная таблица событий.

Обозначение события	Описание частного события	Сигналы на схемах
S_I^t	Ожидание занесения задачи в очередь	*S_task_arrived*
S_{FQ}^t	В очереди присутствуют свободные места	*S_fifo_full*
S_Q^t	Задача в очереди	*S_task_in_queue*
S_{ZPj}^t	Запрос *j*-го процессора диспетчером	*S_proc_zapr_j*
S_{GPj}^t	Задача готова к обслуживанию в *j*-м процессоре, *j*-й процессор помещается в пул занятых	*S_task_ready_j*
S_{PZ}^{pj}	*j*-й процессор выдал сигнал подтверждение запроса	*S_proc_pzapr_j*
S_S^{pj}	*j*-й процессор помещается в пул свободных	*S_proc_free_j*
S_S^p	В процессорном пуле существуют свободные процессоры	*S_proc_free*
S_{PR}^{pj}	Приоритет *j*–го процессора	Реализуется приоритетным блоком
S_{PT}^{pj}	*j*-й процессор принял задачу	*S_proc_prinyal_j*
S_{SL}^{pj}	*j*-й процессор для обслуживания задачи выбран	*S_proc_sel_j*
S_A^{pj}	*j*-й процессор осуществляет обслуживание задачи	
S_{OF}^t	Удалить задачу из очереди	
S_{TO}^{pj}	Снять задачу с выполнения	
S_E^{pj}	Исполнение задачи закончено	
S_{RT}^t	Выдача результата осуществления текущей задачи	
S_O^{pj}	*j*-й процессор прибывает в режиме ожидания	

Основываясь на словесно представленном алгоритме управления процессами и внедренных событий, которые осуществляются в данном алгоритме, система

канонических уравнений, которая описывает данные события, будет обладать таким видом:

- для процесса «клиент», который реализуется диспетчером до момента рандеву:

$$S_I^t(t+1) = x_z \vee S_I^t \overline{S_{FQ}^t}$$

$$S_Q^t(t+1) = S_I^t\, S_{FQ}^t \vee S_Q^t\, \overline{S_{SL}^{pj}};$$

$$S_{ZPj}^t(t+1) = S_Q^t\, S_{SL}^{pj} \vee S_{ZPj}^t\, \overline{S_{PZ}^{pj}}; \qquad (1)$$

$$S_{GPj}^t(t+1) = S_{ZPj}^t\, S_{PZ}^{pj} \vee S_{GPj}^t\, \overline{S_{PT}^{pj}},$$

где: x_z является сигналом запроса задачи на вхождение в очередь.

- для процесса «сервер», который реализуется процессором до момента рандеву:

$$S_S^P(t+1) = (S_S^{p1} \vee S_S^{p2} \vee \ldots \vee S_S^{pN}) \vee S_S^P \overline{S_{ZP}^t}$$

$$S_O^{pj}(t+1) = S_{ZP}^t \vee S_O^{pj} \overline{S_{SL}^{pj}}$$

$$S_{SL}^{pj}(t+1) = \bigvee_{\forall j \in N} S_O^{pj} S_{PR}^{pj} \qquad (2)$$

$$S_{PZ}^{pj}(t+1) = S_{SL}^{pj}\, S_{ZPj}^t \vee S_{PZ}^{pj}\, \overline{S_{GPj}^t};$$

$$S_{PT}^{pj}(t+1) = S_{PZ}^{pj}\, S_{GPj}^t.$$

Система канонических уравнений, которая описывает события после рандеву:

$$S_A^{pj}(t+1) = S_{PT}^{pj}\, S_{GPj}^t \vee S_A^{pj}\, \overline{S_E^{pj}};$$

$$S_{TO}^{pj}(t+1) = S_A^{pj}\, S_E^{pj} \vee S_{TO}^{pj}\, \overline{S_{RT}^t};$$

$$S_{RT}^t(t+1) = S_A^{pj}\, S_E^{pj} \vee S_{RT}^t\, \overline{S_{TO}^{pj}}; \qquad (3)$$

$$S_{OF}^t(t+1) = S_{TO}^{pj}\, S_{RT}^t;$$

$$S_S^{pj}(t+1) = S_{TO}^{pj}\, S_{RT}^t \vee S_S^{pj}\, \overline{S_{GPj}^t}.$$

Уравнениям, непосредственно, соответствует граф недетерминированного автомата (НДА). На рисунке 4 проиллюстрированы графы выбора *j*-го процессора с целью облуживания очередной задачи, а помимо этого клиентской и серверной частей алгоритма управления взаимодействующими процессами до периода «рандеву» (до оператора объединенияJ(&)), а кроме того и часть графа после «рандеву».

Необходимо иметь в виду, что в представленном (аппаратно-ориентированном) алгоритме, функция назначения задач осуществляется диспетчером задач (посетителем). Он устанавливает значение задачи в очереди, наличие свободных процессоров в пуле, создает сигнал запроса процессора, кроме того ожидает ответный сигнал от процессорного блока (сервера), при приобретении которого появляется условие (рандеву), при котором задача направляется на осуществление в определенный выбранный процессор.

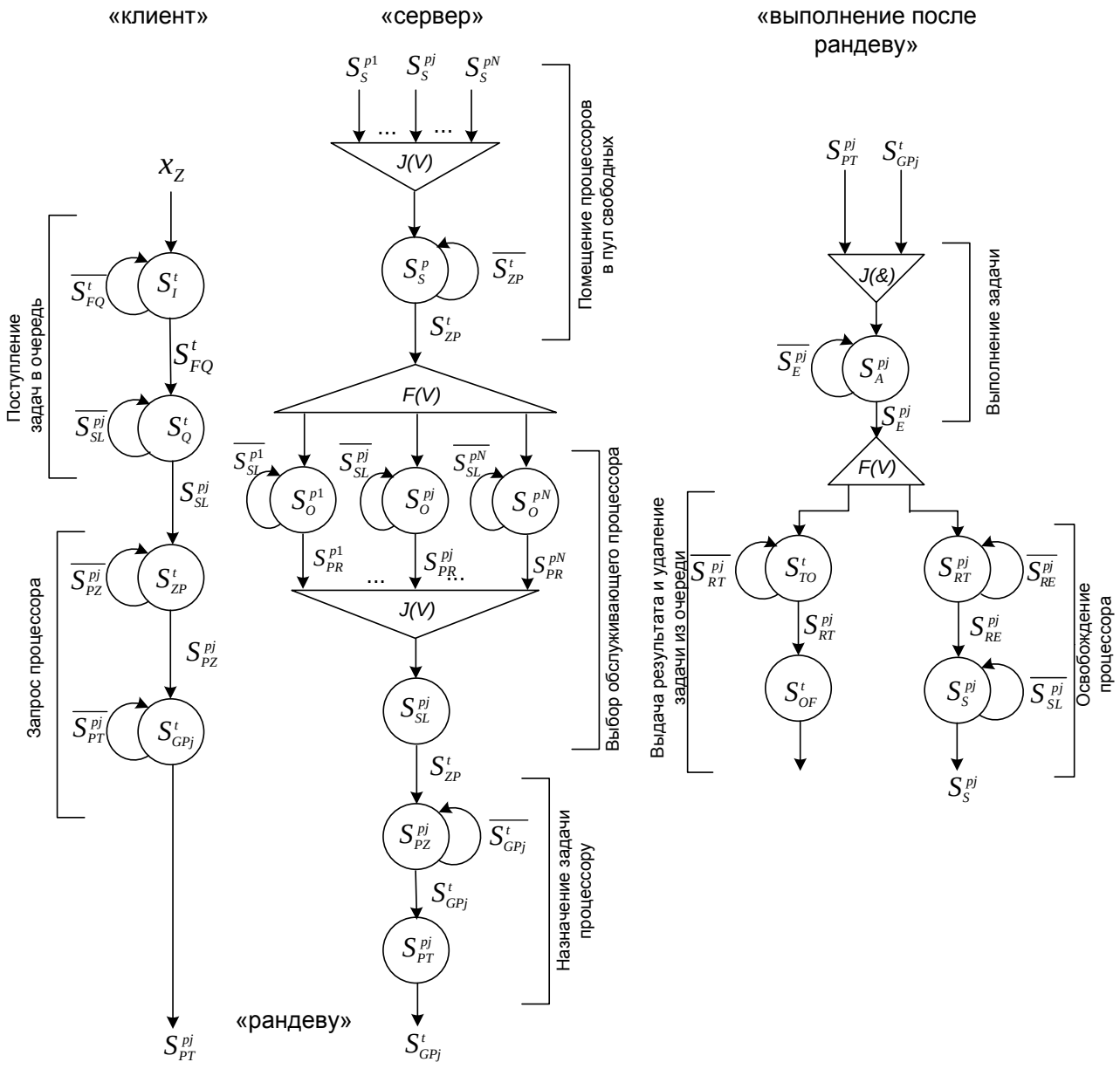


Рис. 5. Граф НДА алгоритма планирования/диспетчеризации задач в многопроцессорной системе

Проиллюстрированная выше система канонических уравнений и граф НДА алгоритма планирования/диспетчеризации с глобальной очередью задач отличается, непосредственно, тем, что здесь обладает местом процедура выбора процессора из всех прибывающих в режиме ожидания. Необходимо иметь в виду, что функция *F(V)* считается разветвительной вершиной, за которой следует осуществление параллельных процессов, направленных на обеспечение состояния ожидания процессоров, которые входят в пул свободных.

Общеизвестно, что представление алгоритма в качестве модели НДА легко осуществляется, кроме того легко преобразуется в программную модель для функционального моделирования на языке VHDL, чему, непосредственно, и посвящен подраздел 1.5.

1.5. Структура аппаратного планировщика/ диспетчера с глобальной очередью задач

Аппаратная поддержка алгоритма диспетчеризации задач осуществлена в несколько упрощенном варианте, а непосредственно, в части постановки задач в очередь, выбора задачи с целью обслуживания в процессоре, помимо этого синхронизации действий серверной части (процессоров), кроме того и клиентской (задачи) на базе метода «рандеву», выбор обслуживающего процессора. Иные функции, к которым относятся ровно как сохранение, так и возобновлении контекста, перемещение из очереди готовых задач в очередь ожидающих и многое другое, осуществляются классическим путем и в представленной модели никаким образом не рассматриваются.

Устройство аппаратной поддержки исследовано, основываясь на примере многопроцессорной системы, которая состоит из восьми процессоров, соединенных общей шиной. Вероятны два варианта подключения диспетчера к процессорному узлу. В первом варианте с целью передачи сигналов между процессором и диспетчером применяется общая шина, что формирует нежелательный дополнительный трафик на ней. В рамках второго варианта устройство диспетчеризации обладает своим интерфейсом взаимодействия абсолютно с каждым процессором, иными словами, в каждом процессорном узле осуществлен интерфейс с целью запроса, а помимо этого и приема идентификаторов задач от диспетчера. Помимо этого, осуществлен второй подход, несмотря на то, что он несколько затрудняет процессорный блок, так как требует дополнительных выводов для снова введенных сигналов. Общий вид системы проиллюстрирован на рис.5.

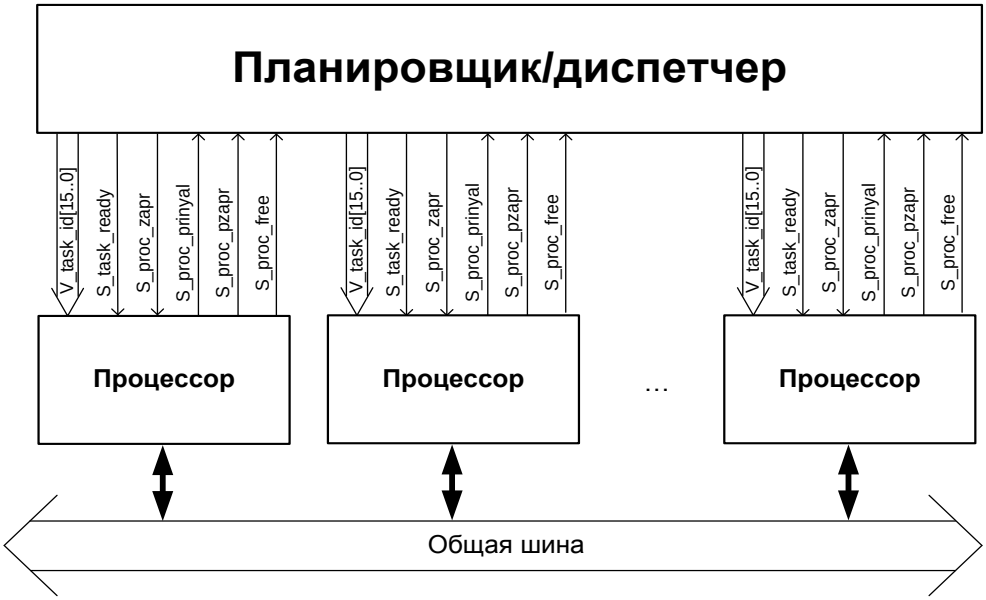


Рисунок 6. Общий вид многопроцессорной системы с аппаратной поддержкой планировщика/диспетчера задач.

К преимуществом представленной организации аналогично можно отнести тот факт, что выход из строя одного из процессоров не воздействует на работоспособность абсолютно всех системы в целом. В случае если выйдет из строя один, либо же несколько процессоров, то работоспособность многопроцессорной системы сохраниться при некотором уменьшении общей производительности.

Интерфейс взаимодействия процессора с диспетчером представлен на рисунке 7.

Рисунок 7. Интерфейс взаимодействия одного процессора с планировщиком/диспетчером.

Клиентский процесс формируется таким образом: если существует определенный процесс, который ожидает обслуживания, то диспетчер при условии, что существуют свободные процессоры, определяет активный уровень сигнала «Запрос» *(S_proc_zapr)* на линии того процессора, который выбирается из числа свободных согласно заданному алгоритму. Процессор, который получил сигнал диспетчера о его выборе с целью обслуживания текущей задачи, покидает спящий режим, а после сообщат диспетчеру о том, что готов к приему и осуществлению задачи. Непосредственно, для этого, процессор определяет активный уровень сигнала «Подтверждение запаса» *(S_proc_pzapr)*, который воспринимает диспетчер. Помимо этого, процессор снимает сигнал «Свободен» *(S_proc_free)*, и, следовательно, выходит из пула свободных. Процессор назначен, и после, диспетчер переходит к прямой передаче идентификатора задачи процессору. Для этого он, непосредственно, извлекает идентификатор задачи из внутреннего регистра, а кроме того выставляет его на шину *(V_task_id)*, и перенаправляет в процессор сигнал «Процесс готов» *(S_task_ready)*. Процессор воспринимает активную степень сигнала «Процесс готов», тем самым ему обеспечивается, что сведения на шине являются достоверными *(V_task_id)*, иными словами, на шине имеется идентификатор текущей задачи. Процессор закрепляет представленные сведения в собственном внутреннем регистре, и когда его деятельность с шиной будет завершена, он выставляет сигнал «Процесс принята» *(S_proc_prinyal).*

Серверный процесс формируется следующим образом. Процессор, который не занят обслуживанием задачи (свободный), прибывает в спящем режиме. Сигнал «Свободен» *(S_proc_free)*, который поступает от процессора к диспетчеру, принимает, в данном случае активный уровень. В случае если в очереди существуют готовые задачи, то выбирается обслуживающий процессор согласно приоритетному алгоритму, при этом первый приоритет в представленном случае присвоен процессору, обладающему номером 1. Выбранный процессор выходит из спящего режима, кроме того выбирает задачу из начала очереди *(S_fifo_read)*, а

после приобретает идентификатор задачи по шине *(V_task_id)*, и выставляет сигнал «Процесс принял» *(S_proc_prinyal)*, а затем переходит в режим «Занято».

После вышеперечисленного, планировщик/диспетчер способен снять сведения с шины и перейти к назначению следующей задачи. Процессор, который приобрел идентификатор задачи, переходит, непосредственно, к ее обслуживанию. Абсолютно все сигналы интерфейса сняты (на всех линиях фиксируется пассивный уровень). По итогу работы, процессор уведомит диспетчера о том, что он свободен, посредством установления активного уровня сигнала «Свободен» *(S_proc_free)*, тем самым опять окажется в пуле свободных процессоров, а помимо этого диспетчер снова будет принимать во внимание его в процессе распределения задач.

С функциональной очки зрения, устройство планирования/диспетчеризации задачи представлено в качестве четырех блоков (рисунок 8), к которым относятся следующие:

- блок управления очередью задач;
- блок очереди задач FIFO;
- блок выбора процессора с целью обслуживания очередной задачи;
- блок синхронизации и назначения задач согласно процессорам.

Блок управления очередью задач необходим для приема идентификаторов задач в систему. Он осуществляет анализ на наличие свободных мест в очереди задач, и в случае если свободные места имеются, то помещает идентификатор новой задачи в блок очереди *FIFO*. Помимо этого, реализует выборку идентификатора задачи из очереди согласно принципу «первым пришел – первым обслужился», а затем помещает его в свободный процессор.

Следовательно, можно сказать, что блок реализует часть функций планировщика, иными словами создает очередь задач и отслеживает ее состояние, применяя для этого два сигнала. Активный уровень сигнала «*FIFO* заполнено» *(S_fifo_full)* уведомляет планировщика, непосредственно, о том, что принимать новые задачи запрещается, поскольку отсутствуют свободные регистры для осуществления записи, а кроме того и хранения новых идентификаторов. Аналогичным образом дело обстоит и с поступающими процессами. В случае если

в очереди отсутствуют места для осуществления записи новых идентификаторов задач, то они планировщиком никаким образом не воспринимаются. Что касаемо второго сигнала - «*FIFO* пустое» *(S_fifo_empty)*, то он указывает на то, что абсолютно ни один идентификатор в очередь не был поставлен, иными словами, очередь задач пуста. О наличии задачи в очереди говорит сигнал *S_task_in_queue*. Основываясь на нем, формируется решение о том, выводить из спящего режима процессор, либо же нет. В случае если очередь задач является пустой, то блок синхронизации не обращается к процессорам. Сигналы «*FIFO* заполнено» *(S_fifo_full)*, а кроме того и «*FIFO*пустое» *(S_fifo_empty)* создаются и поступают в блок управления очередью задач из блока *FIFO*.

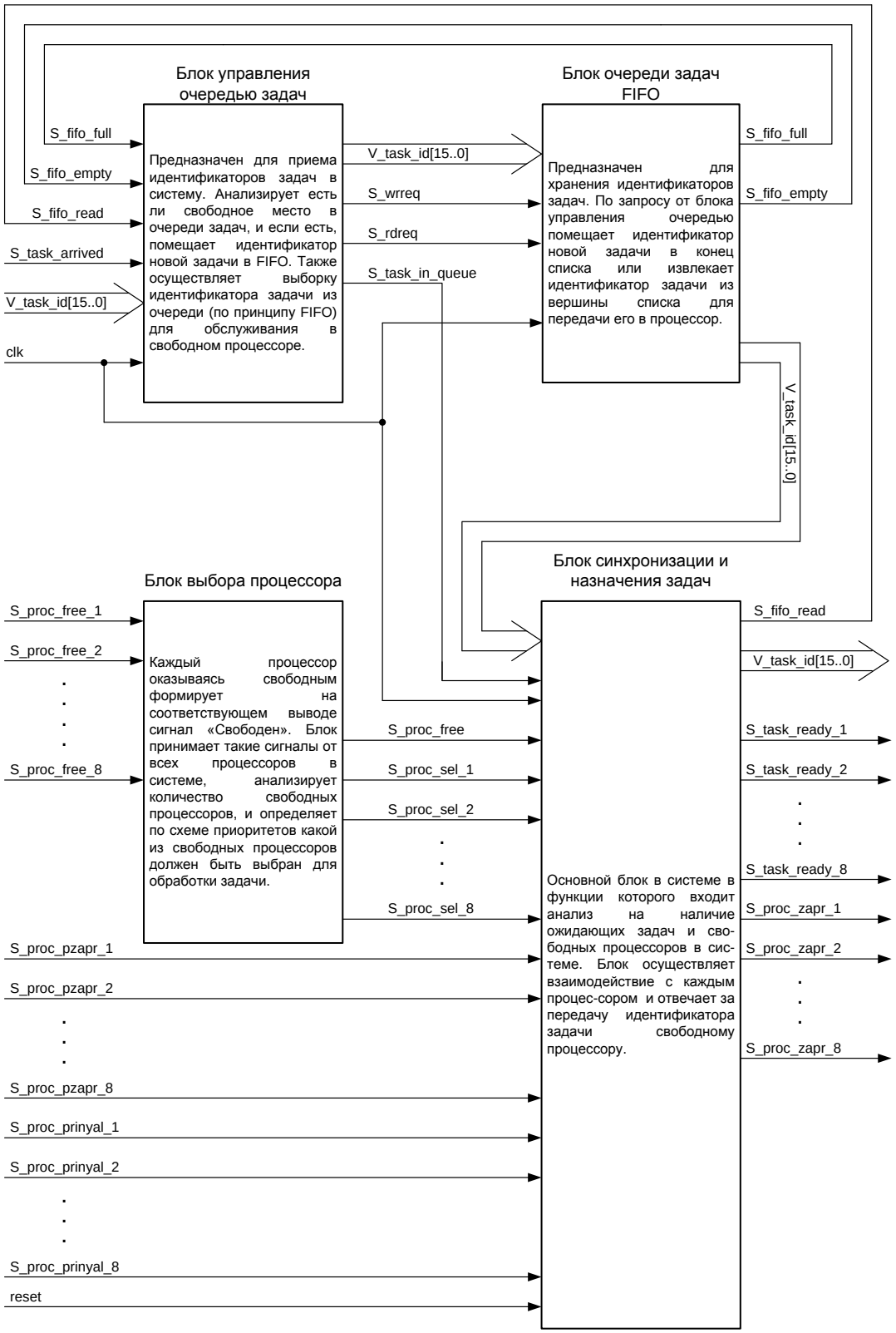


Рисунок 8. Структура устройства диспетчеризации задач.

В блоке управления очередью задач существует шина данных *(V_task_id)*, посредством которой передаются идентификаторы задач. О том что, на шине имеется новый процесс, планировщика уведомляет нарастающий фронт сигнала

«Поступил процесс» *(S_task_arrived)*. В случае если очередь не является заполненной, то планировщик принимает сведения с шины *V_task_id* и записывает их данные в очередь *FIFO*, в конец списка. Если очередь *FIFO* является заполненной, то сведения на входной шине игнорируются. Запрос очередной задачи осуществляет один из процессоров, который выбран в виде обслуживающего. Для указанных целей применяется сигнал «Запрос задачи» *(S_fifo_read_j)*. Основываясь на указанном сигнале, блок диспетчеризации создает сигнал «Чтение *FIFO*» *(S_fifo_read)* и перенаправляет его блоку управления очередью задач.

Планировщик управляет блоком очереди задач *FIFO*, прибегая к помощи сигналами *S_wrreq*, а помимо этого и *S_rdreq*.*S_wrreq* применяется с целью записи в очередь *FIFO*. А *S_rdreq*, в свою очередь, для чтения из нее.

Блок очереди задач *FIFO* осуществляет хранение поступающих идентификаторов задач. Идентификатор представляет собой часть контекста задачи, и способен состоять из нескольких полей, одно из которых указывает на адрес задачи в памяти многопроцессорной системы. Согласно запросу от блока управления очередью заносит идентификатор новой задачи, непосредственно, в конец списка, либо же извлекает идентификатор запускаемой задачи из вершины списка с целью передачи его в процессор.

Блок выбора процессора принимает абсолютно все сигналы «Свободен» *(S_proc_free)* от всех процессоров в системе. Он устанавливает, имеются ли в настоящий момент в системе свободные процессоры. В случае если в наличии существует хотя бы один процессор, который не занят обслуживанием текущей задачи, то данный блок определяет активный уровень сигнала «Есть свободные процессоры» *(S_proc_free)* и именно подобным образом уведомляет блок управления назначением задач о том, что существуют «спящие» процессоры.

Выбор процессоров реализуется схемой взаимоисключения, а кроме того и схемой приоритетов. С целью сообщения блоку диспетчеризации о том, непосредственно, какой из процессоров выбран с целью обслуживания, служат выводы *(S_proc_sel_1,……,S_proc_sel_8)*.

Блок синхронизации и назначения задач исследует сведения о наличии ожидающих задач в очереди, а помимо этого и свободных процессоров, которые назначаются с целью обработки данных задач. Указанный блок, применяя механизм синхронизации «рандеву», реализует взаимодействие очереди задач с процессорами согласно описанному выше интерфейсу. Блок выбора процессора создает сведения о том, какой из свободных процессоров выбран с целью обслуживания текущей задачи. Идентификатор принятой на обслуживание задачи передается блоком очереди задач *FIFO* по шине*V_task_id*. Помимо этого, блок управления очередью задач уведомляет о состоянии очереди и реализует выборку идентификатора задач из блока очереди *FIFO* согласно требованию блока синхронизации и назначения задач.

Моделирование на языке VHDL и анализ результатов

Модель устройства планировщика/диспетчера была осуществлена на языке *VHDL* в качестве 4-х программных модулей. Первый модуль осуществляет логические функции первого блока – управление очередью задач. Второй модуль представляет собой блок FIFO. А третий модуль является блоком управления свободными процессорами, а что касаемо четвертого модуля, то это блок синхронизации. В виде примера далее представлен фрагмент программы для осуществления блока управления очередью:

```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.all;
3.  USE ieee.std_logic_unsigned.all;
4.  USE ieee.std_logic_arith.all;
5.  ENTITY block_upr_ocher IS
6.  PORT
7.  (
8.  CLK             : IN STD_LOGIC;
9.  V_task_id       : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
10. S_task_arrived  : IN STD_LOGIC;
11. S_fifo_full     : IN STD_LOGIC;
12. S_fifo_empty    : IN STD_LOGIC;
13. S_fifo_read     : IN STD_LOGIC;
14. V_fifo_data_o   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
15. S_wrreq         : OUT STD_LOGIC;
16. S_rdreq         : OUT STD_LOGIC;
17. S_task_in_queue : OUT STD_LOGIC
18. );
19. END block_upr_ocher;
```

```
20. ARCHITECTURE block_upr_ocher_arch OF block_upr_ocher IS
21. SIGNAL reg_task_id      : STD_LOGIC_VECTOR(15 DOWNTO 0);
22. SIGNAL S_task_arrived1  : STD_LOGIC;
23. SIGNAL S_task_arrived2  : STD_LOGIC;
24. BEGIN
25. PROCESS
26. BEGIN
27. WAIT UNTIL CLK='1';
28. S_task_in_queue<=not S_fifo_empty;
29. reg_task_id<=V_task_id;
30. S_task_arrived1<=S_task_arrived;
31. S_task_arrived2<=S_task_arrived1;
32.
33. IF S_task_arrived1='0' AND S_task_arrived2='1' THEN
34.       IF S_fifo_full='0' THEN
35.             S_wrreq<='1';
36.             V_fifo_data_o<=reg_task_id;
37.       END IF;
38. ELSE
39.       S_wrreq<='0';
40. END IF;
41. IF S_fifo_read='1' THEN
42.       S_rdreq<='1';
43. ELSE
44.       S_rdreq<='0';
45. END IF;
46. END PROCESS;
47. END block_upr_ocher_arch;
```

В заголовке (а именно, строки 5-19) отмечаются входные и выходные переменные. Их имена соответствуют описанным ранее сигналам для блока управления очередью задач. Непосредственное осуществление логических выражений исполнено в строках 28-45, которые включаются в состав процесса (строки 25-46), который задает работу синхронных схем согласно переднему фронту синхросигнала *CLK*. Указанный модуль применялся, непосредственно, как элемент в модели устройства диспетчера, результаты изучения которого будут представлены далее.

Ввод схем и работа осуществлялась в свободно распространяемой версии системы *Xilinx ISE WebPACK Design Suite* 12.1, а что касаемо моделирования, то оно осуществлялось в той же системе.

Перед тем как перейти к моделированию требуется описать еще два дополнительно созданных блока, которые не считаются частью диспетчера, однако входящих в состав макета и использовавшихся на стадии моделирования.

Блок генерации задач. В процессе моделирования требуется задавать диспетчеру идентификаторы задач, которые готовы к исполнению. Задачи поступают в систему регулярно, и их число способно являться очень большим. Следовательно, на входе блока управления очередью задач устанавливается блок генерации задач, который посредством равных промежутков времени выставляет новый идентификатор задачи, а помимо этого самостоятельно уведомляет об этом блок управления очередью, тем самым имитируя поток задач. Для того чтобы запустить работу блока генерации задач вполне достаточно подать активный уровень на вход блока start.

Блок процессора. Помимо генератора задач дополнительно был сформирован блок имитации работы процессора, который в полной мере поддерживает интерфейс «процессор-диспетчер». Когда идентификатор задачи приобретен блоком имитации процессора, то запускается счетчик, который осуществляет отсчет нескольких десятков тактов (значение представляется возможным задать любое). В данное время указанный блок отключается от диспетчера, абсолютно аналогично, если бы реальный процессор, приобретя задачу, перешел бы к ее обслуживанию. Счетчик отсчитывает фактическое время, которое затрачивается процессором на обработку задачи, после чего переходит в пул свободных, а помимо этого уведомляет об этом диспетчера.

Общий вид структурной схемы модели диспетчера задач в составе восьмипроцессорной системы проиллюстрирован на рисунок 9.

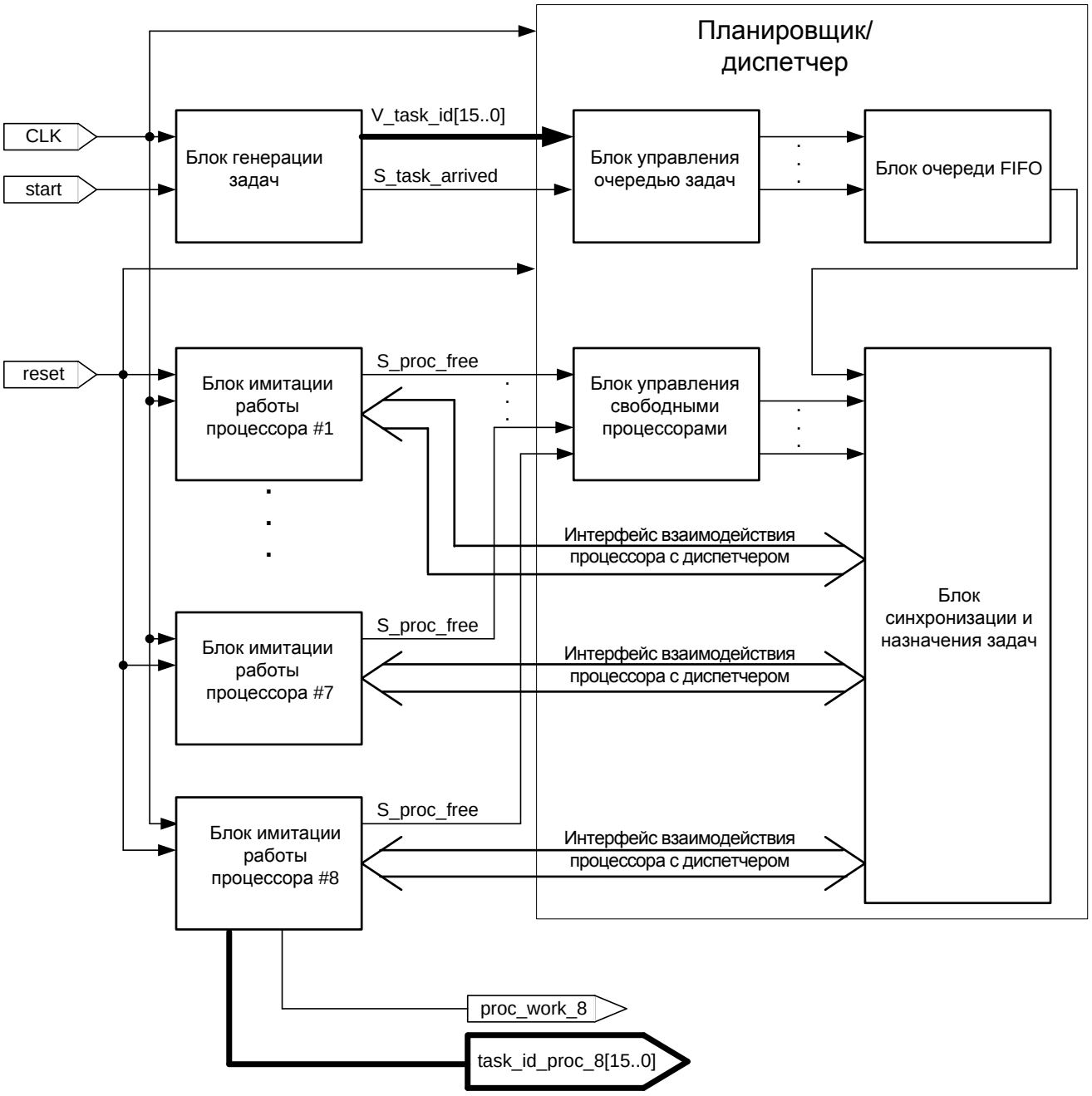


Рисунок 9. Схема проведения экспериментальных исследований

Почти все представленные выше сигналы для указанной системы считаются внутренними и генерируются самостоятельно. На входе системы подаются синхросигналы *CLK*, непосредственно, от которого тактируются абсолютно все блоки, сигналы начального сброса системы (*reset*), а помимо этого и запуска генератора задач (*start*).

С целью проверки результатов моделирования применялись выводы *proc_work_[j]* и *task_id_proc_[j][15..0]*. Что касаемо первого вывода (*proc_work_[j]*), то он свидетельствует о том, что *j*–й процессор приобрел

идентификатор текущей задачи, а помимо этого приступил к ее обслуживанию. Шина *task_id_proc_[j][15..0]* перенаправляет сведения о том, какой идентификатор приорел *j*–й процессор. С целью контроля поступающих в систему идентификаторов задач вносились сведения о состоянии шины данных блока очереди *FIFO*. Указанные состояния на временной диаграмме обозначены, непосредственно, как T*ask_id_F*. Результаты моделирования проиллюстрированы на рис.9.

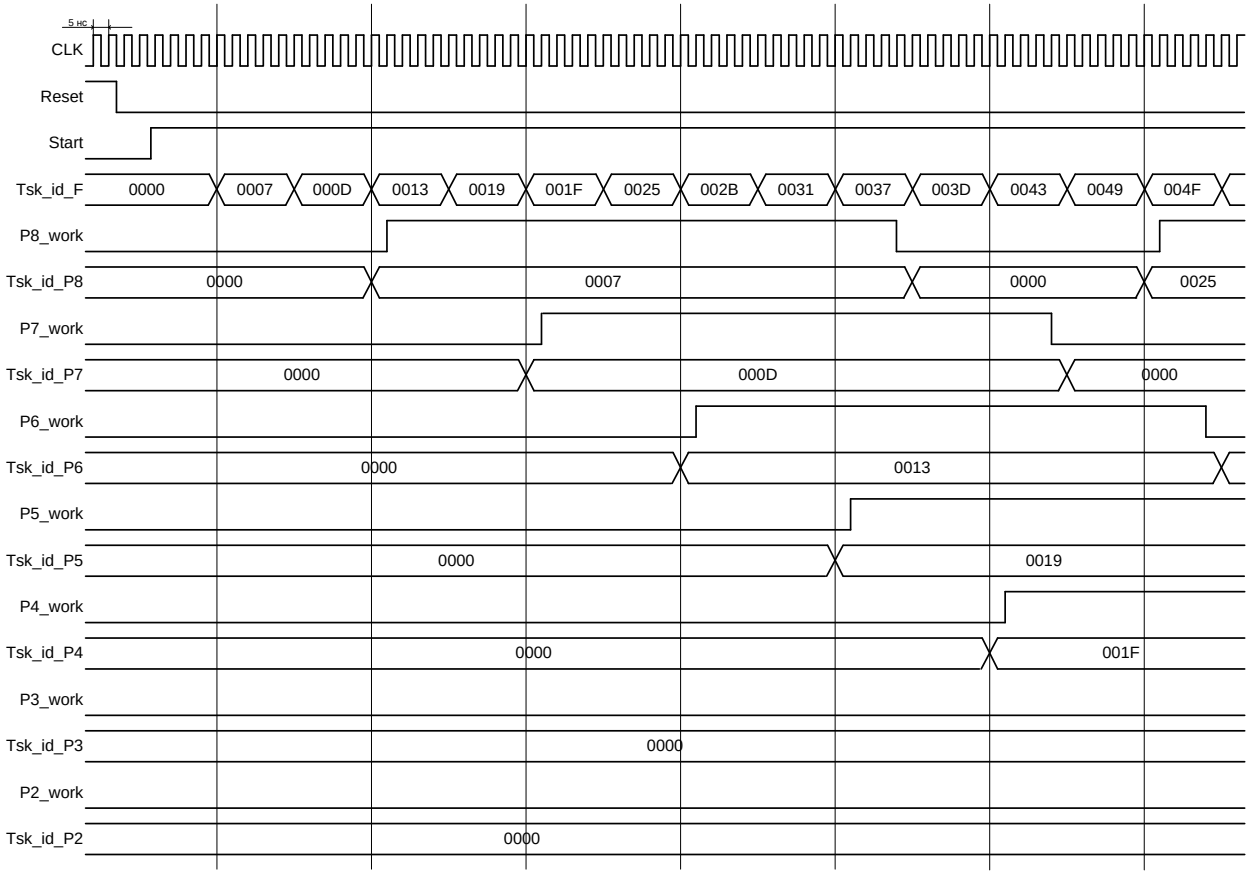


Рисунок 10. Временные диаграммы работы аппаратного диспетчера с глобальной очередью.

Из временной диаграммы можно наблюдать, как идентификатор первой поступившей задачи (*«0007» hex*) ровно через десять тактов, непосредственно, поступил в процессор под номером 8, после чего процессор выставил сигнал о том, что занят обслуживанием. Поскольку восьмой процессор оказался занятым, то второй идентификатор задачи (*«000D» hex*) поступил через десять тактов в процессор, который обладает номером семь. Данный процессор, так же как и предыдущий, выставил сигнал о том, что является занятым и приступил к обслуживанию второй задачи.

Следовательно, является очевидным, что те десять тактов, предшествующих назначению задачи процессору уходят на синхронизацию процессов.

Выводы к первой главе.

Выполнена формализация алгоритмов планирования/диспетчеризации с распределенной очереди задач многопроцессорной системы. Формализация базируется на логике недетерминированных автоматов, что, непосредственно, обеспечивает надежность, а кроме того и безопасность многопроцессорных операционных систем, а помимо этого простоту их аппаратной реализации.

Основываясь на НДА – моделей сформирована VHDL – модель аппаратного планировщика/диспетчера задач.

Произведено моделирование алгоритма планирования/диспетчеризации задач, который осуществлен на языке VHDL. Приобретены временные характеристики аппаратной реализации, которые демонстрируют, что латентность (иными словами, задержка исполнения задачи в обстоятельствах бесконфликтности) является в сотни раз ниже традиционной программной реализации. Согласно предварительным оценкам использование аппаратного планировщика/диспетчера предоставляет возможность повысить производительность многопроцессорной системы порядка на 10-20%.

2. Аналитическое моделирование планировщика/диспетчера задач

Достоинством функционального моделирования является получение характеристик планировщика и диспетчера задач близких к реальным. Однако ряд вероятностно-временных характеристик, таких как время ответа (отклика), загрузка процессорных узлов и диспетчера задач, количество задач в очередях, время ожидания задач в очередях и других функциональное моделирование не дает. Чтобы получить такие характеристики необходимо провести аналитическое моделирование, основанное на системах и сетях массового обслуживания.

2.1 Анализ методов планирования и диспетчеризации задач современных операционных систем

Имеется огромное количество методов диспетчеризации (помимо этого встречается термин «дисциплина диспетчеризации»), разделяющиеся на два огромных класса, такие как: бесприоритетные и приоритетные, согласно которым создается очередь готовых к исполнению задач. В процессе бесприориетной диспетчеризации выбор задачи осуществляется в заранее определенном порядке без учета ее значимости и времени обслуживания. В процессе осуществления приоритетных методов диспетчеризации определенным задачам оказывается преимущественное право попасть в состояние исполнения. В собственную очередь, приоритеты способны являться фиксированными (считаются величиной постоянной абсолютно на всем цикле задачи), либо же динамическими (меняются в зависимости от определенных условий согласно конкретным правилам). Зафиксированные приоритеты осуществляются несколько проще, а для осуществления динамических требуются дополнительные расходы, однако их применение подразумевает наиболее справедливое распределение процессорного времени между процессами. Ряд методов диспетчеризации, а кроме того и их классификация проиллюстрированы на рис.10.

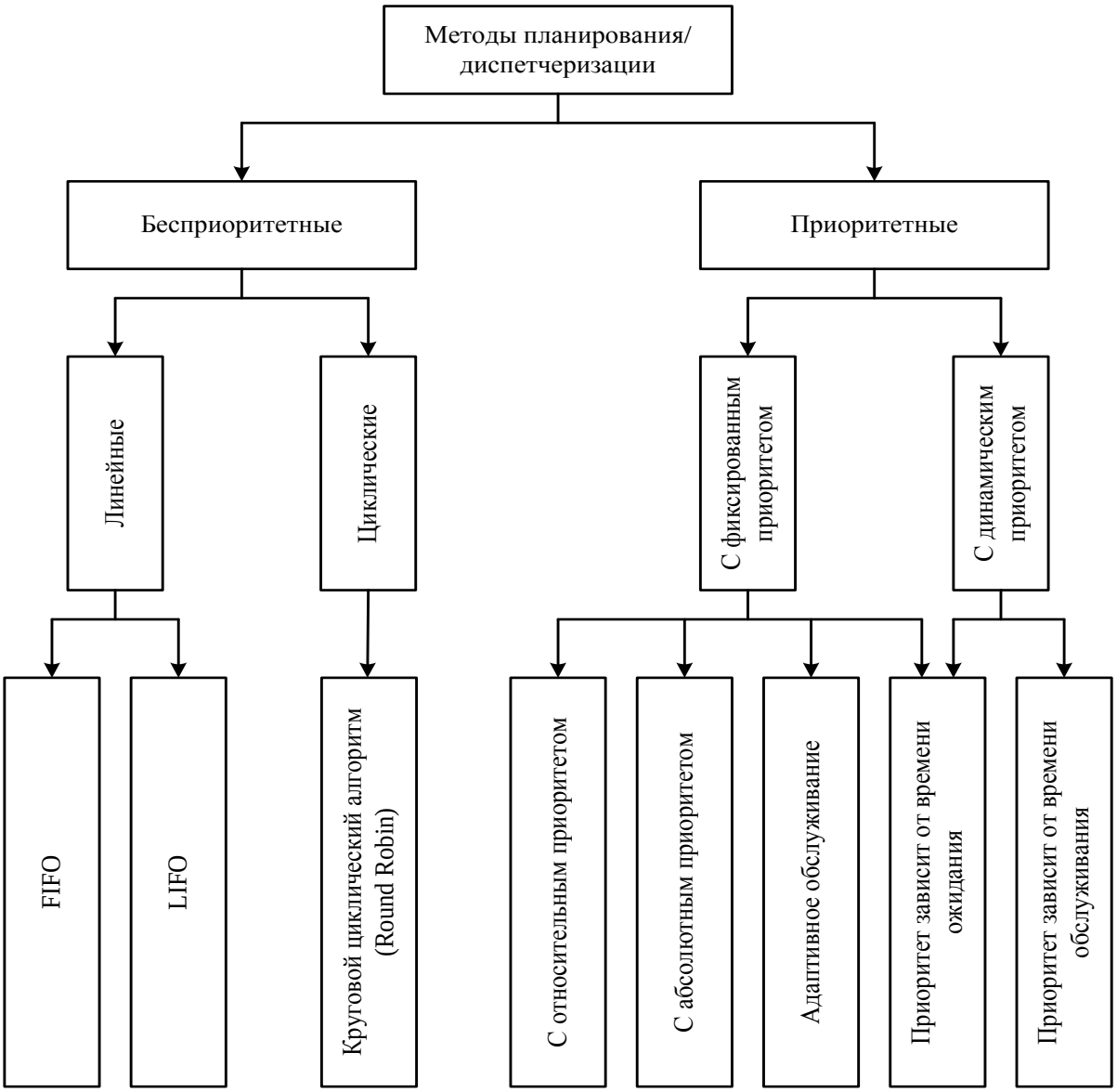


Рисунок 11 Методы планирования/диспетчеризации, применяемые в операционных системах

Метод FIFO – «первый пришел – первый обслужился». Модель представленного метода проиллюстрирована на рисунке 12. Новая задача поступает, непосредственно, в конец очереди. Первой обслуживается та заявка, которая прибывает в начале очереди. Используется для задач, которые получают полноценное обслуживание, когда задача, которая захватила процессор не освобождает его до того момента, пока она полноценно не исполнится. Оптимизация планирования, в данном методе, отсутствует.

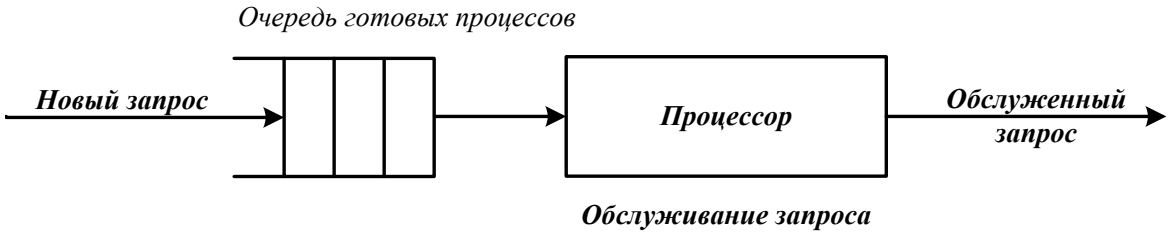


Рисунок 12. Модель планирования/диспетчеризации *FIFO*

Усовершенствованная FIFO представляет собой модифицированный метод, в соответствии с которым, те задачи, которые были заблокированы в ходе работы (попали в состояние блокировки в ядре ОС по причине занятости определенного ресурса, к которому задача обратилась в ходе обработки), по завершении перехода в состояние готовности, ставятся в очередь готовых задач перед задачами, которые до этого еще не исполнялись. Представленный подход предоставляет возможность осуществить стратегию обслуживания «по возможности завершать вычисления согласно порядку их появления».

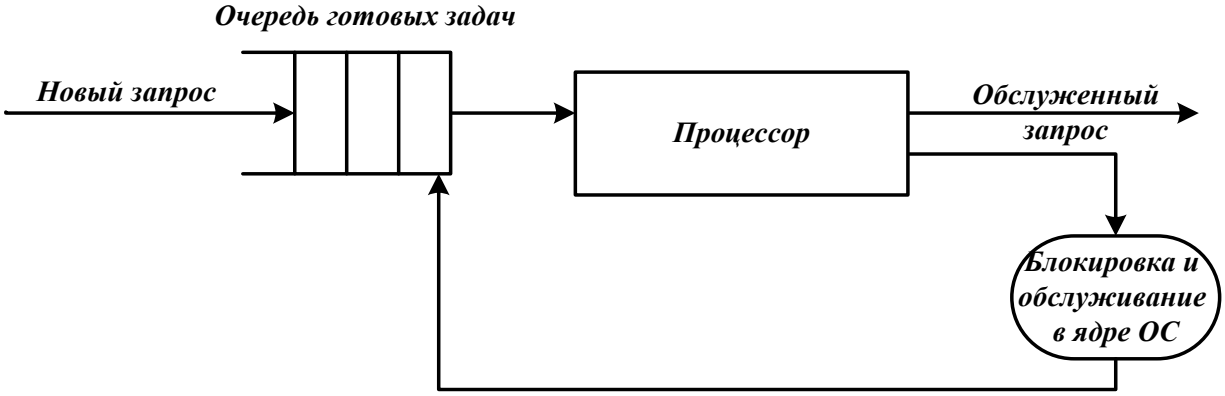


Рисунок 13. Модель планирования/диспетчеризации «модифицированный *FIFO»*

В качестве достоинств, представленных моделей можно отнести простоту осуществления и малые затраты системных ресурсов на создание очереди задач. Общий недостаток представленных методов состоит в следующем: при повышении загрузки ПРОЦЕССОР возрастает среднее время ожидания обслуживания, следовательно, при этом короткие задачи вынуждены ждать такое же время, сколько и трудоемкие.

Метод LIFO – «последний пришел – первый обслужился». Отличие данного метода, как и *FIFO*, состоит в простоте реализации. Модель данного метода проиллюстрирована на рис.14.

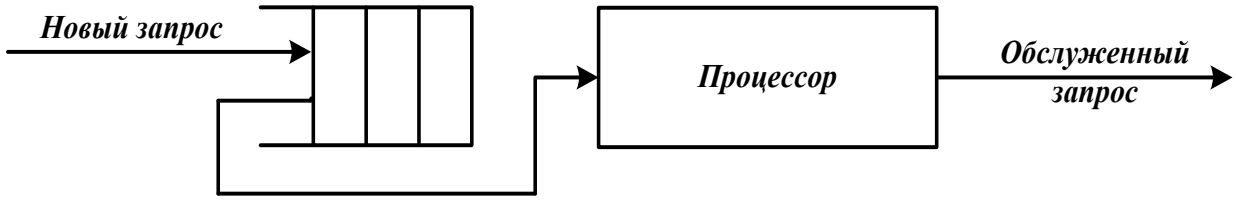


Рисунок 14. Модель планирования/диспетчеризации *LIFO*

Первой обслуживается та заявка, которая прибывает в конце очереди. Используется для задач, которые получили полноценное обслуживание, когда задача, которая захватила процессор не освобождает его до того момента, пока она полноценно не исполнится. Оптимизация планирования, в данном методе, отсутствует.

Среднее время ожидания в очереди, при некоторой определенной интенсивности поступления запросов, а помимо этого и интенсивности их обслуживания считается одинаковым, вне зависимости от характеристик осуществляемых процессов. Иными словами, продолжительные небольшие, согласно времени, процессы, будут ожидать в очереди, практически одинаково, что, в свою очередь, приводит к повышению среднего времени обслуживания запросов.

Метод Round Robin (RR) представляет собой «циклическое обслуживание», в основе которого заложен метод *FIFO*. Но период обслуживания абсолютно каждого процесса ограничен и обуславливается квантом времени τ_k . В случае если i–я задача из очереди успевает исполняться за определенный интервал τ_k , то, следовательно, она освобождает процессорный узел. В случае если же не успевает, то ее обслуживание прерывается, и она помещается в конец очереди. Упрощенная модель проиллюстрирована на рисунок 15. Метод приобрел обширное распространение при осуществлении стратегии разделения процессорного времени.

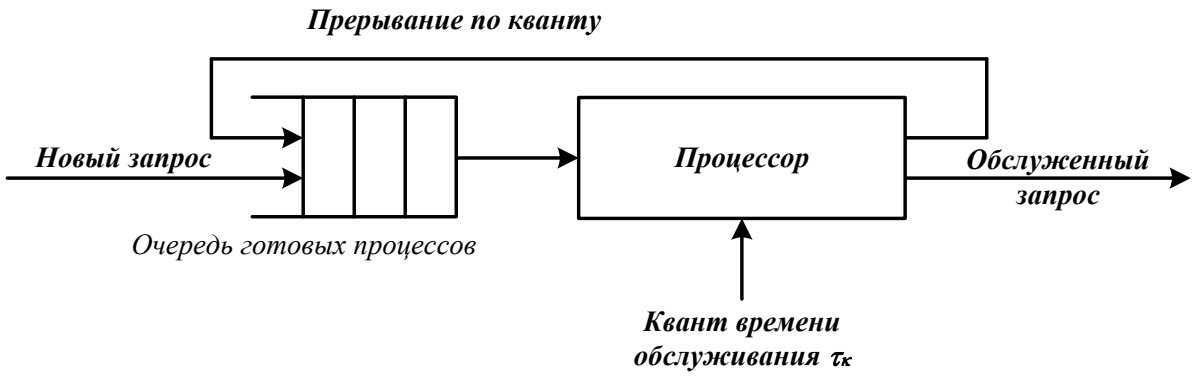


Рисунок 15. Модель метода планирования/диспетчеризации *RR*

Короткие задачи, как правило, обслуживаются стремительнее, обладают наименьшим средним временем ожидания в системе, нежели длинные задачи. Уровень благоприятствования коротким задачам тем больше, насколько меньше продолжительность кванта мультиплексирована, а помимо этого насколько ближе она к продолжительности интервала номинального применения ресурса процессом. Но сокращение продолжительности кванта ведет к повышению задержек, которые требуются для отработки прерываний, а кроме того и перераспределения ресурса. Это происходит по причине увеличения частоты прерываний, что в особенности отрицательно способно отразиться на отработке «длинных» запросов. Именно поэтому, на практике, применяют всевозможные модификации представленного алгоритма, к примеру: SJN и SRT. SJN («Кротчайшая задача следующая»). Необходимо, чтобы абсолютно для каждой задачи была известна оценка в потребностях машинного времени. Представленный метод диспетчеризации подразумевает наличие одной очереди задач, которые готовы к исполнению. Задачи, которые в процессе собственного выполнения были временно заблокированы, снова попадают в конец очереди готовых к исполнению, вместе с вновь поступающими. Это, в свою очередь, приводит к тому, что условия, которым необходимо немного времени с целью собственного завершения, вынуждены ожидать ПРОЦЕССОР, вместе с продолжительными задачами, что далеко не всегда хорошо. С целью ликвидации данного недостатка был предложен метод *SRT* («Следующая задача требует наименьшее время для своего завершения»).

Метод диспетчеризации с относительными приоритетами. Представленный метод базируется на приоритетах. Задача не способна быть вытеснена иными требованиями. Очередь готовых задач является одна. Новая задача абсолютно всегда ставиться в конец очереди. На выполнение выбирается условие с большим приоритетом. Смена исполняющегося задания осуществляется в таких случаях, как: а) процесс обслужился; б) процесс перешел в состояние блокировки в ядре ОС. Схема метода диспетчеризации с относительными приоритетами проиллюстрирована на рис.16.

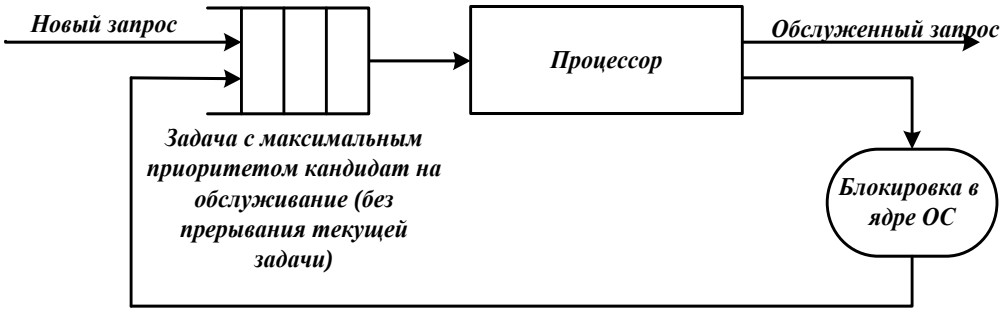


Рисунок 16. Модель метода планирования/диспетчеризации с относительными приоритетами

В качестве достоинств можно отнести учет приоритетности задач.

А в виде недостатков следующие:

1) вероятность бесконечного откладывания обслуживания;

2) потеря контроля над системой, когда управление, непосредственно, передается задаче.

Метод планирования/диспетчеризации с абсолютными приоритетами. В представленном методе задача способна быть вытеснена требованием с несколько высшим приоритетом. Вновь поступившие задачи ставятся в конец единственной очереди, из которых выбирается условие с предельным приоритетом. Смена исполняющегося задания осуществляется в случаях, когда: а) процесс завершен; б) в очереди возник процесс с наибольшим приоритетом. Схема метода диспетчеризации с абсолютными приоритетами проиллюстрирована на рис.17.

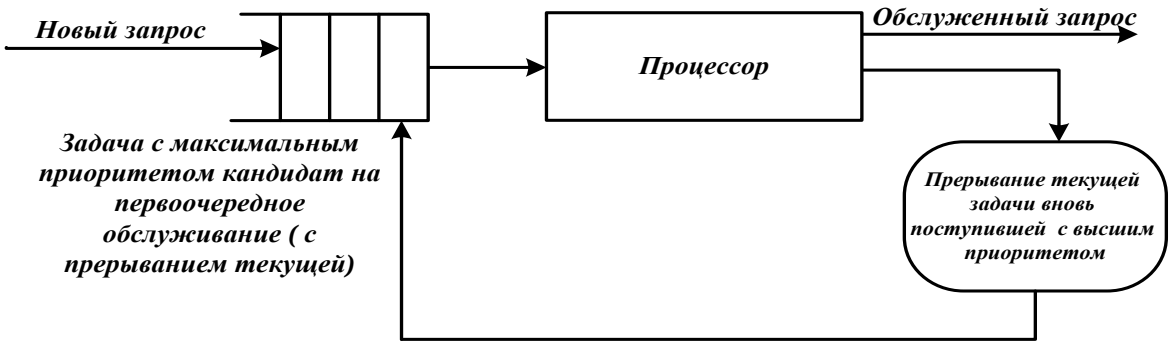


Рисунок.17. Модель метода планирования/диспетчеризации с абсолютными приоритетами

В качестве достоинств представленной модели выделяют следующие:

1) принимается во внимание приоритетность задач;

2) не утрачивается контроль над системой (любой квант конечен).

В качестве недостатков представленной модели выделяют следующие:

1) вероятность бесконечного откладывания обслуживания низкоприоритетных задач;

2) достаточно трудная реализация, так как довольно трудно установить момент для перерасчета приоритетов.

Метод адаптивного обслуживания. Алгоритм базируется на квантовании, однако величина кванта прибывает в зависимости от приоритета: насколько выше приоритет, настолько больший период времени задача сможет занимать ПРОЦЕССОР. Выбор из очереди способен реализовываться согласно первому компоненту, либо же согласно приоритету. Схема метода адаптивного обслуживания проиллюстрирована на рисунок 18.

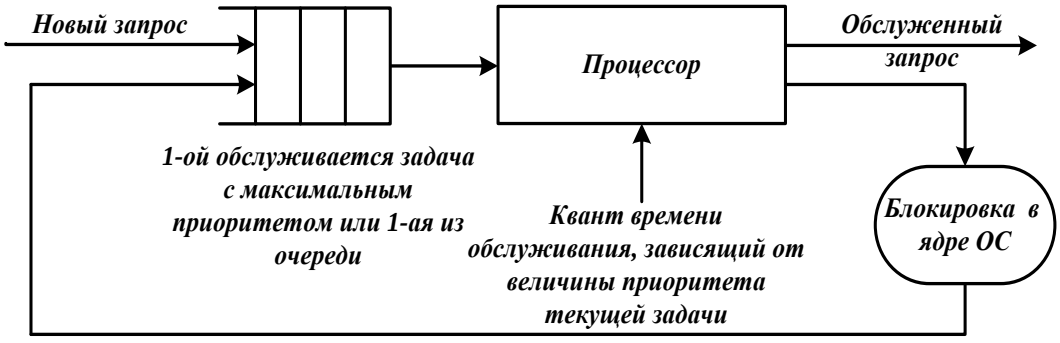


Рисунок 18. Модель метода адаптивного обслуживания

В качестве достоинств представленной модели выделяют следующие:

а) принимает во внимание приоритетность задач;

б) простота осуществления, так как отсутствует пересчет приоритетов;

в) отсутствует полная потеря контроля над системой, поскольку любой квант является конечным.

Недостаток представленной модели заключается в том, что задача не способна быть вытолкнула иной, пока не завершится квант.

Метод диспетчеризации с приоритетом, зависящем от времени обслуживания. Метод базируется на абсолютных приоритетах. В процессе осуществления задачи ее приоритет сокращается с каждым тиком, либо квантом. В случае если приоритет текущей задачи становится несколько меньше приоритета задачи, которая стоит в очереди готовых к выполнению, она будет вытеснена с исполнения. Это предоставляет возможность сократить дискриминацию задач, которые возникают при применении метода диспетчеризации с абсолютными приоритетами. Смена исполняющегося задания осуществляется в таких случаях, как: а) задача завершена; б) задача перешла в состояние блокировки, а помимо этого и обслуживания в ядре ОС; в) приоритет задачи становится несколько меньше, нежели у ожидающего в очереди готовых к выполнению задач с большим приоритетом; г) в очереди возникает задача с наибольшим приоритетом. Схема метода проиллюстрирована на рисунок 19.

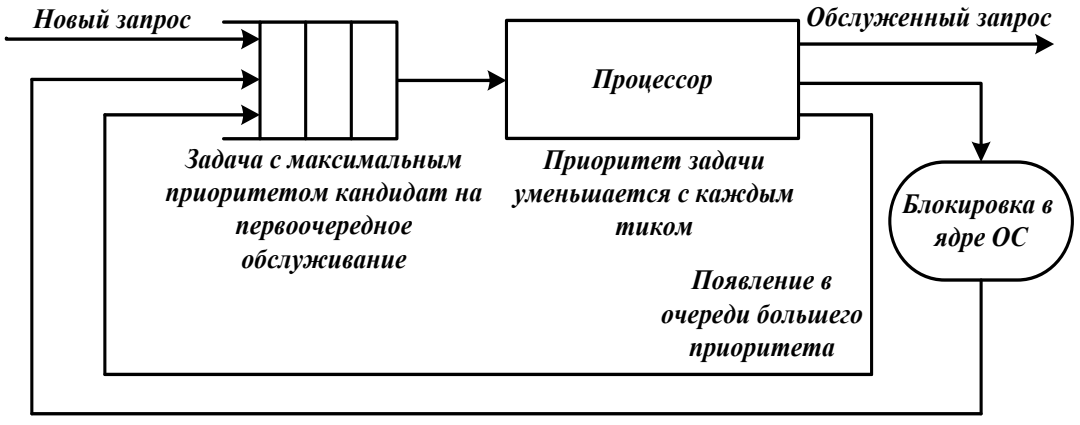


Рисунок 19. Модель метода планирования/диспетчеризации с приоритетами, зависящими от времени исполнения

В качестве достоинств представленной модели выделяют следующие:

а) принимает во внимание приоритетность задач;

б) сокращает возможность недобросовестного применения механизмов приоритетов.

В качестве недостатков представленной модели выделяют следующие:

а) возможность постоянного откладывания низкоприоритетных задач;

б) достаточно сложная организация, поскольку требуется пересчет приоритетов.

Метод диспетчеризации с приоритетом, который, непосредственно, зависит от времени ожидания в очереди готовых процессов. Согласно мере ожидания в очереди готовых к выполнению задач, приоритет процесса возрастает с каждым типом, либо же квантом. В случае если приоритет исполняющегося задания становится меньше приоритета требования, который стоит в очереди готовых, то оно будет вытеснено с исполнения. Это предоставляет возможность исключить дискриминацию задач, которая возникает в процессе использования методов диспетчеризации с абсолютными приоритетами, а помимо этого и методов диспетчеризации с приоритетами, которые зависят от времени исполнения. Смена исполняющегося задания осуществляется в таких случаях, как: а) задача завершена; б) задача перешла в состояние блокировки, а кроме того и обслуживания в ядре ОС; в) приоритет исполняющегося задания становится меньше, нежели у ожидающего в очереди готовых на выполнение заданий, задания с большим приоритетом; г) в очереди возник процесс с наибольшим приоритетом. Схема метода диспетчеризации с приоритетами, которые зависят от времени ожидания проиллюстрирована на рисунок 20.

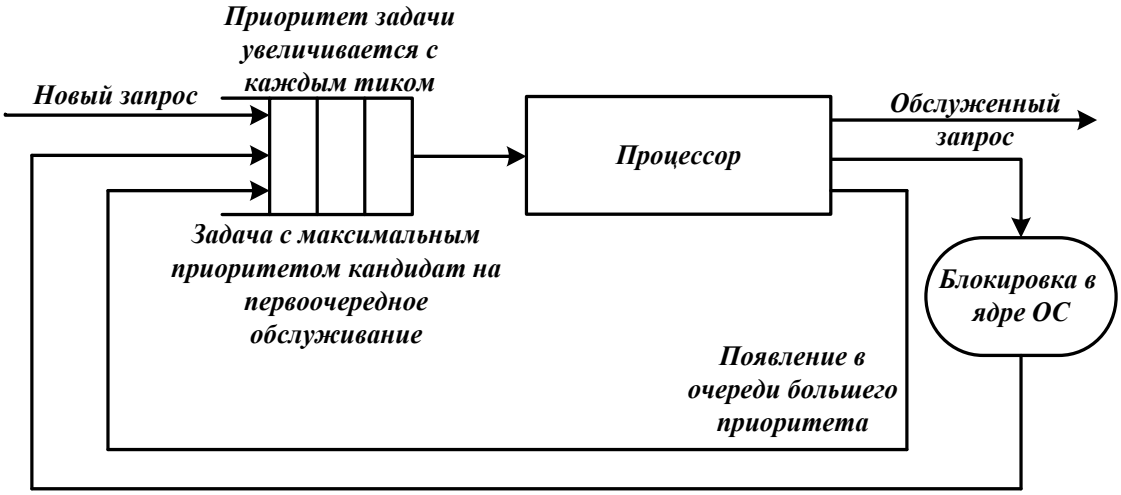


Рисунок 20. Модель метода планирования/диспетчеризации с приоритетами, зависящими от времени ожидания

В качестве достоинства можно выделить то, что данная модель учитывает приоритетность задач.

В качестве недостатков представленной модели выделяют следующие:

1) возможность постоянного откладывания низкоприоритетных задач;

2) сложная организация, поскольку требуется пересчет приоритетов.

2.2 Аналитическое моделирование диспетчеров задач с однородным входящим потоком задач, бесприоритетной очередью с ограничением числа мест

В реальных ВС требования на обслуживание поступают, как показывает практика, в случайные моменты времени. Распределение времен поступления, а помимо этого и обслуживания способно подчиняться конкретным законам. В представленном подразделе математическое моделирование системы осуществляется на базе СМО типа *M*/*M*/1, по причине того, что это, непосредственно, экспоненциальная СМО (поток поступающих задач считается простейшим, продолжительность обслуживания распределена согласно экспоненциальному закону). При формировании модели реальной ВС, на базе СМО *M*/*M*/1, а кроме того и изучении ее будут приобретены максимальные характеристики рассматриваемой системы. Подобная модель предоставляет худший вариант поведения ВС, при котором приобретают верхние граничные вероятностно-временные характеристики, а кроме того и гарантию

работоспособности моделируемой вычислительной системы при заданных параметрах.

Математическая модель ДИСПЕТЧЕР ЗАДАЧ, которая обладает стратегией распределенной диспетчеризацией (индивидуальными диспетчерами) сформирована из *n* одноканальных СМО ($S_1,...,S_n$) (рис.20.). Абсолютно каждая СМО моделирует обслуживание в подсистеме «диспетчер задач - процессорный узел» («ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР»). Источник S_0 моделирует потоки задач λ_0, а помимо этого поглощает обслуженные задачи (иными словами, выдача результата пользователю). Перед СМО $S_1, S_3,..., S_{n-3}, S_{n-1}$ создаются очереди, которые обладают ограничением количества мест k. В случае если существует свободное место в одной из очередей, то задача занимает его и прибывает в локальной очереди до того момента, пока не поступил на исполнение в ПРОЦЕССОР. В случае если применяется режим квантования, то незаконченная задача по завершении текущего кванта назначается ДИСПЕТЧЕР ЗАДАЧ на дообслуживание, а помимо этого помещается в конец той самой очереди, где она прибывала до этого, в противном случае, результат выдается пользователю, а в локальной очереди освобождается одно место. По итогу исполнения задачи ДИСПЕТЧЕР ЗАДАЧ просматривает собственную очередь. В случае если в ней существуют требования на обслуживание, то назначается на исполнение задача, которая стоит в голове списка. В случае если очередь процессорста, система «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» переходит в режим ожидания, то «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» в представленном случае осуществляет балансировку загрузки согласно определенному алгоритму. По этой причине при переполнении i–й очереди (которая обладает вероятностью $p_{откi}$) задачи извлекаются из нее, а после передаются некоторой вероятностью в менее загруженную j–ю очередь.

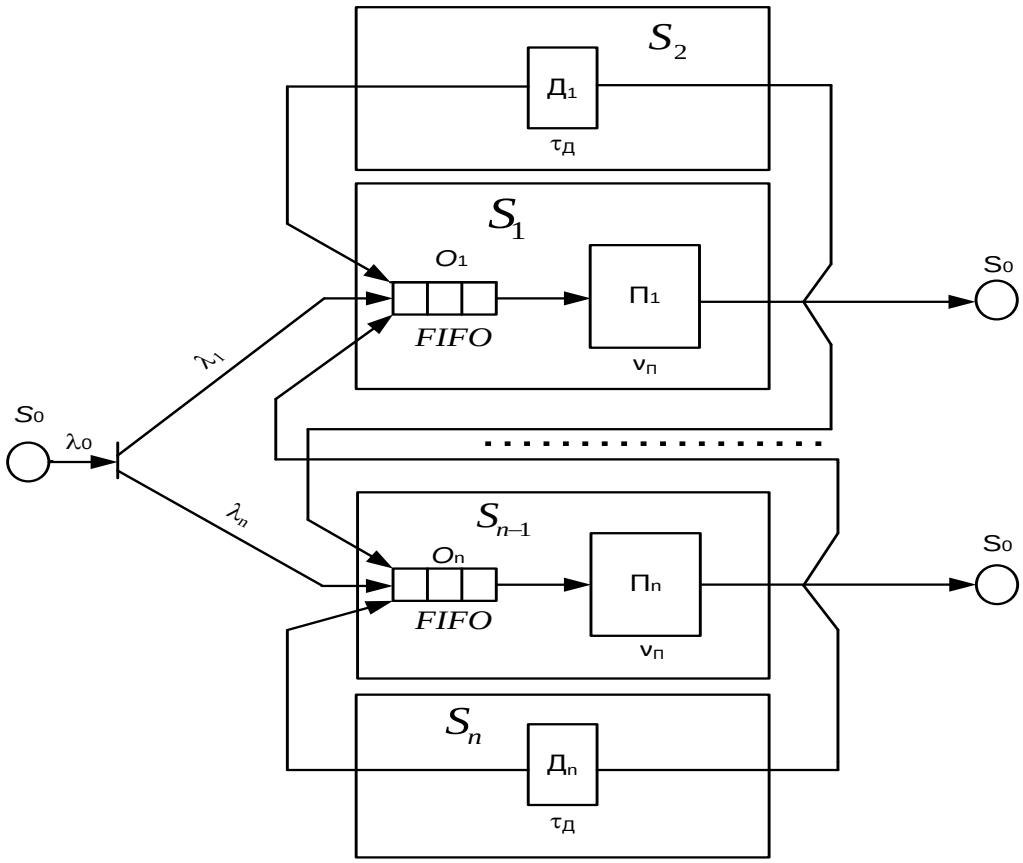


Рисунок 21. Модель многопроцессорной системы с распределенным планировщиком/диспетчером задач

Рассматриваемая система приведена в качестве СеМО, которая является комплексом одноканальных СМО. На вход абсолютно каждой СМО ($S_1, S_3, ..., S_{n-3}, S_{n-1}$) поступает поток запросов, в общем случае со следующей интенсивностью $\lambda_1, \lambda_3, ..., \lambda_{n-3}, \lambda_{n-1}$, $\lambda_0 = \sum_{i=1}^{n-1} \lambda_i$. Интенсивность обслуживания подсистемой «Диспетчер – Процессор» потока требований равняется $\frac{1}{(\tau_Д + \nu_П)}$. Задача, которая поступила в момент, когда подсистема является, занята, становится перед ней в очередь, а помимо этого ожидает обслуживания. Допроцессорстим вне зависимости от того, какое количество задач поступает в очередь, является не способной вместить свыше k задач, из которых одна обслуживается, а k–1 только ожидают. Условия, которые не попали в очередь ожидания Oi, обслуживаются в ином месте, а кроме того направляются в иную очередь. Следует рассмотреть

интенсивности потоков задач в исследуемой МПС. В общем случае их можно представить такой матрицей вероятности, как:

$$\left|\begin{array}{c|cccccccc}
 & S_0 & S_1 & S_2 & S_3 & S_4 & \dots & S_{n-1} & S_n \\
S_0 & & p_{01} & & p_{03} & & \dots & p_{0(n-1)} & \\
S_1 & p_{10} & & p_{12} & p_{13} & & \dots & p_{1(n-1)} & \\
S_2 & & p_{21} & & & & \dots & & \\
S_3 & p_{30} & p_{31} & & & p_{34} & \dots & p_{3(n-1)} & \\
S_4 & & & & p_{43} & & \dots & & \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
S_{n-1} & p_{(n-1)0} & p_{(n-1)1} & & p_{(n-1)3} & & \dots & & p_{(n-1)n} \\
S_n & & & & & & \dots & p_{n(n-1)} &
\end{array}\right| \quad (4)$$

Система уравнений интенсивностей задач внутри СеМО представлена следующим образом:

$$\begin{cases}
\lambda_0 = P_{10}\lambda_1 + P_{30}\lambda_3 + \dots + P_{(n-1)0}\lambda_{n-1}; \\
\lambda_1 = P_{01}\lambda_0 + P_{21}\lambda_2 + P_{31}\lambda_3 + \dots + P_{(n-1)1}\lambda_{n-1}; \\
\lambda_2 = P_{12}\lambda_1; \\
\lambda_3 = P_{03}\lambda_0 + P_{13}\lambda_1 + P_{43}\lambda_4 + \dots + P_{(n-1)3}\lambda_{n-1}; \\
\lambda_4 = P_{34}\lambda_3; \\
\dots\dots\dots

$$\alpha_i = \frac{\lambda_i + \sum\limits_{\forall n (n \neq i)} p_{ij} \cdot \lambda_i}{\lambda_0}, \quad i,\ j = \overline{1,n}. \qquad (6)$$

Задача приобретает отказ только в том случае, когда подсистема «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» является занятой и все *k* места в очереди – также, при всем при этом вероятность отказа

$$p_{отк_i} = \frac{\rho_i^{k+1}(1-\rho_i)}{1-\rho_i^{k+2}}, \qquad (7)$$

где $\rho_i = \lambda_i \cdot (\tau_Д + \nu_П)$.

По выражениям и выкладкам, приобретем формулу с целью нахождения средней длины очереди перед системой «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР»

$$l_i = \frac{\rho_i^2 \cdot (1-\rho_i) \cdot (1-\rho_i^k \cdot (k+1-k \cdot \rho_i))}{(1-\rho_i^{k+2}) \cdot (1-\rho_i)^2} = \frac{\rho_i^2 \cdot (1-\rho_i^k \cdot (k+1-k \cdot \rho_i))}{(1-\rho^{k+2}) \cdot (1-\rho_i)}. \qquad (8)$$

Далее установим среднее время ожидания задачи в очереди w_i. Условие поступает в систему в конкретный момент времени. С вероятностью p_0 система «Диспетчер – Процессор» является свободной и период ожидания равен нулю. С вероятностью p_1 задача поступит в СМО, перед ней будет отсутствовать очередь, и, следовательно, она будет ожидать своего обслуживания на протяжении времени $\tau_Д + \nu_П$ (среднее время обслуживания одной задачи). А что касаемо вероятности p_2 в очереди, то перед рассматриваемой задачей будет располагаться еще одна и период ожидания в среднем будет следующим: $2 \cdot (\tau_Д + \nu_П)$, и так далее. При $q = k+1$, иными словами, когда снова приходящая задача застанет «Диспетчер – Процессор» занятым еще k задач в очереди, то период ожидания в представленном случае аналогично равняется нулю, поскольку задача не принимается в указанную очередь. Средний период задержки задачи в i–й очереди равняется:

$$w_i = \frac{l_i}{\lambda_i}. \qquad (9)$$

Средний период ожидания задачи в очередях сети рассчитывается в соответствии со следующим выражением:

$$W = \sum_{i=1}^{n-1} \alpha_i \cdot w_i , \qquad (10)$$

где α_i – является коэффициентом передач; w_i – является временем ожидания в очереди перед i-й СМО.

Время ответа в системе с распределенным диспетчером устанавливается согласно следующей формуле:

$$U = \sum_{i=1}^{n-1} \alpha_i \cdot u_i \qquad (11)$$

где α_i – является коэффициентом передач; u_i – является временем ответа i-й подсистемы «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР».

2.3 Аналитическое моделирование планировщиков/диспетчеров задач с неоднородным потоком задач

Порой в реальных ВС (в особенности в системах реального времени – СРВ) входящие задачи на обслуживание подразумевают под собой неоднородный поток, который содержит условия различных типов (различная интенсивность поступления в систему, различные классы приоритетов). К примеру, в СРВ на исполнение способны поступать задачи различного приоритета (различной степени значимости). Высокоприоритетные требования будут обслуживаться стремительнее, поскольку требуется как можно быстрее отреагировать на них и выделить результат. Задачи с наиболее низким приоритетом (иными словами, низкой значимостью) в случае, если у поступившей задачи абсолютный приоритет в сопоставлении с текущей задачей, прерываются. После, по завершении обслуживания прервавшего требования, дообслуживатеся прерванная задача (в случае если, безусловно, за период обработки высокоприоритетной задачи в систему не поступило условие с приоритетом больше, нежели у прерванной задачи). Прерванная (снятая с исполнения) задача переводится в режим ожидания. В случае если поступившая задача обладает относительным приоритетом в сопоставлении с исполняемой, она дожидается завершения работы с текущим

заданием. Отталкиваясь от представленного ранее, рассмотрим МПС с распределенным диспетчером задач, непосредственно, как систему с неоднородным потоком задач на обслуживание.

Предположим, что представленная система не считается экспоненциальной (то есть поступает простейший поток задач, а продолжительность обслуживания его распределяется согласно произвольному закону), в таком случае она представляется как СМО M/G/1. При формировании модели ВС на основе СМО M/G/1 и изучении ее будут приобретены характеристики рассматриваемой системы, которые приближены к реально возможным. Следовательно, принята к изучению система, которая основана на СМО типа M/G/1. На вход абсолютно каждой СМО ($S_1, ..., S_{n-1}$)поступает процессор экспоненциальный поток задач с интенсивностью $\lambda_1, ..., \lambda_{n-1}$, $\lambda_0 = \sum_{i=1}^{n-1} \lambda_i$. Средний период обслуживания системой «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» потока требований равняется $\left(\tau_Д + \nu_П\right)$ и распределяется согласно конкретному закону. Задача, которая поступила в момент, когда система является занятой, становится перед ней в очередь, которая ограниченна числом мест k, и ждет обслуживания. Требования, которые не попали в очередь ожиданияOi, с вероятностью отказа направляются в иные очереди.

Значения интенсивностей потока, который распределяется согласно n очередям, ($\lambda_1, \lambda_3, ..., \lambda_{n-3}, \lambda_{n-1}$) находятся аналогично (4-5) и прибывают в зависимости от входящей интенсивности потока задач источника S_0 (λ_0), а помимо этого и вероятностей переходов из СМО S_i в СМО S_j. Так как в $S_1, S_3, ..., S_{n-1}$ способны появляться отказы в обслуживании по причине переполнения очередей (с вероятностью $P_{отк_i} = \sum_{\forall n[n \neq j]} p_{i,j}$), то, следовательно, интенсивности $\lambda_1, \lambda_3, ..., \lambda_{n-3}, \lambda_{n-1}$ падают. Но представленное падение возмещается переназначением задач из заполненных очередей. В соответствии с выражениями (6) и (7) устанавливаются коэффициенты передач α_i, а помимо этого и

вероятности отказа в обслуживании i–й СМО ($P_{откi}$). Обслуживание поступившей задачи в СМО $S_1,...,S_{n-1}$ реализуется согласно произвольному закону распределения B(x) с математическим ожиданием $\bar{x}$.

Обслуживание с относительными приоритетами в подсистеме «диспетчер задач – процессорный узел». Исследуем систему с n процессорами, которые поступают на i–ю СМО с относительными приоритетами. Источник S_0 моделирует неоднородный поток задач на обслуживание (λ_0), который распределяется согласно n-1 СМО ($\sum_{i=1}^{n-1}\lambda_i = \lambda_0,\ i = 1,3,...,n-1$, λ_i – является интенсивностью потока задач на каждую СМО), а помимо этого и поглощает обслуженные задачи. Поток требований на обслуживание состоит, непосредственно, из H классов задач (подразумевается H приоритетов) $\lambda_i = \lambda_{j1} + \lambda_{j2} + ... + \lambda_{jH}$. Перед i–м процессором создается очередь задач, в которую поступают неоднородные требования. Принятая на обслуживание задача прибывает в очереди до того момента, пока не поступит на исполнение в подсистему «Диспетчер – Процессор». СеМО сформировано из n одноканальных СМО (рис.22).

В процессе обслуживания задач с относительными приоритетами, они исполняются подсистемой «Диспетчер-Процессор» в соответствии с порядком поступления. В случае прихода в очередь наиболее приоритетного требования, оно будет обслужено вне очереди, после завершения работы с текущей задачей.

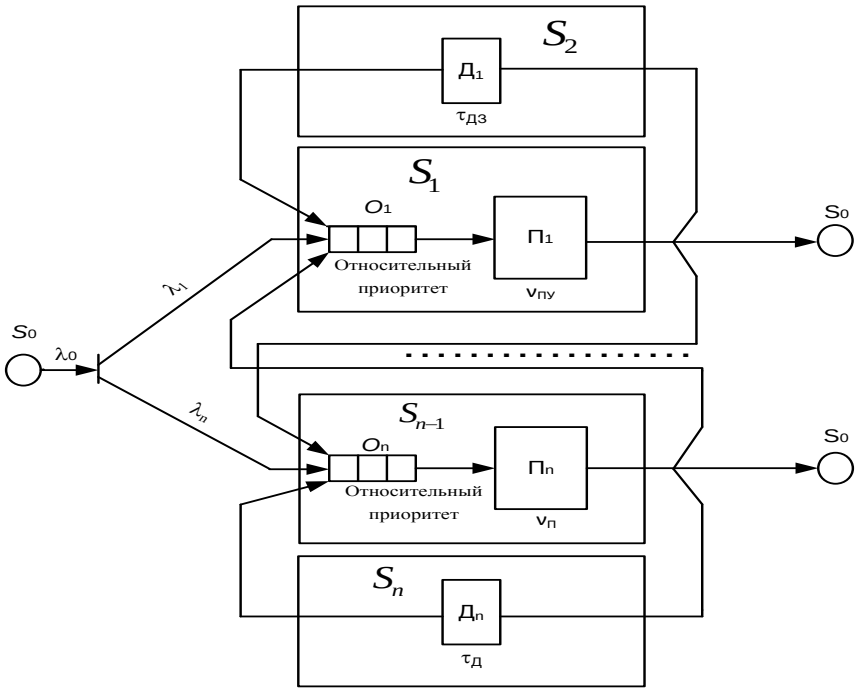


Рисунок 22. Схема многопроцессорной системы с индивидуальными диспетчерами задач и обслуживанием с относительными приоритетами

Средняя продолжительность обслуживания системой «Диспетчер-Процессор» потока требований равняется $\tau_Д + \nu_П$. Задача, которая поступила в момент, когда система занята, становится перед ней в очередь, которая ограничена количеством мест *k*, а помимо этого ждет обслуживания. Требования, которые не попали в очередь ожидания *Oi,* с вероятностью отказа направляются в иные очереди. Значения интенсивностей потока, который распределяется согласно n-очередям, ($\lambda_1, \lambda_3, ..., \lambda_{n-3}, \lambda_{n-1}$) находится аналогично (4-5) и прибывают в зависимости от входящей интенсивности потока задач источника S_0 (λ_0), количества типов задач и вероятностей переходов из СМО S_i в СМО S_j.В соответствии с выражениями 6 и 7 устанавливаются коэффициенты передач α_i, а помимо этого и вероятности отказа в обслуживании i–й СМО ($P_{откi}$).

Помимо этого можно заменить систему M/G/1 с ограниченной очередью на такую же систему с неограниченной очередью. СеМО, которая состоит и СМО с неограниченными очередями задач, намного проще проанализировать и выделить основные характеристики, нежели которая состоит из СМО с ограничением длины

очереди в том случае, когда очередь содержит огромное количество мест (не меньше 16ти для задач СРВ) в подобном случае погрешность вычислений составит порядка не меньше 2,5%. Средний период ожидания задач с относительными приоритетами в очереди перед i–й подсистемой «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» определиться согласно следующему выражению:

$$w_k^{ОП} = \frac{\sum_{i=1}^{H} \lambda_i \cdot t_i^2 \cdot (1 + \nu_{t_i}^2)}{2 \cdot (1 - \sum_{i=1}^{k-1} \rho_i) \cdot (1 - \sum_{i=1}^{k} \rho_i)} \quad (k = 1, ..., H) \qquad (12)$$

где ρ_i – является коэффициентом загрузки системы ($\rho_i = \frac{\lambda_i}{\mu_i}$, λ_i – является интенсивностью поступления задач i-го приоритета; μ_i – является интенсивностью обслуживания задач i-го приоритета), t_i – является средним временем обслуживания задач i-го приоритета ($t_i = \frac{1}{\mu_i}$), ν_{t_i} – является коэффициентом вариации продолжительности обслуживания задачи i-го приоритета.

Рассчитаем средний период ожидания абсолютно всех приоритетных задач во всех очередях исследуемой системы:

$$W = \sum_{i=1}^{n-1} \alpha_i \cdot \sum_{k=1}^{H} w_k^{ОП}, \; i=1, 3, \ldots, n\text{-}3, n\text{-}1. \qquad (13)$$

Продолжительность ответа системы ДИСПЕТЧЕР ЗАДАЧ, а помимо этого и стратегией распределенной очереди при неоднородном входящем потоке с относительными приоритетами составит следующее:

$$U = \sum_{i=1}^{n} \alpha_i \cdot u_i, \qquad (14)$$

где $u_i = W_i^{ОП} + \tau_{обслі}$, $W_i^{ОП} = \sum_{i=1}^{k} w_i^{ОП}$, $\tau_{обслі} = t_Д + \nu_{Пі}$.

Обслуживание с абсолютными приоритетами в подсистеме «диспетчер задач – процессорный узел». Далее рассмотрим n-процессорную систему с абсолютными приоритетами. Поток условий на обслуживание состоит, непосредственно, как и в первом случае с относительными приоритетами, из H-

классов задач $\lambda_i = \lambda_{j1} + \lambda_{j2} + ... + \lambda_{jH}$. Перед i–й подсистемой «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» создается очередь задач, в которую поступают разнородные условия. Принятая на обслуживание, задача прибывает в локальной очереди до того момента, пока не поступит на исполнение в ПРОЦЕССОР. СеМО состоит, непосредственно, из n-одноканальных СМО (рис.23).

В случае обслуживания с абсолютными приоритетами, задачи исполняются согласно порядку поступления, но в случае поступления в очередь наиболее приоритетного запроса на исполнение, он будет обслужен вне очереди, прервав, при этом, исполнение текущей задачи.

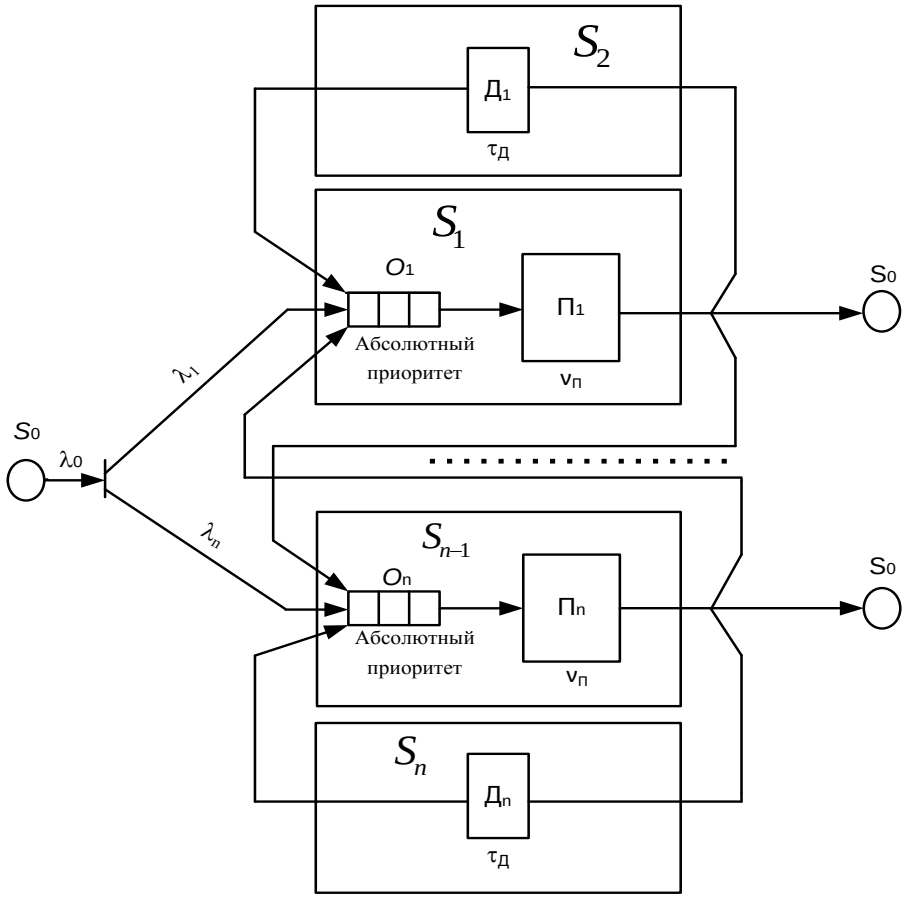


Рисунок 23. Схема многопроцессорной системы с индивидуальными диспетчерами задач и обслуживанием с абсолютными приоритетами

На вход очереди подсистемы «Диспетчер-Процессор» поступает поток из H-типов задач на исполнение со следующими интенсивностями $\lambda_i = \lambda_{j1} + \lambda_{j2} + ... + \lambda_{jH}$, где H является приоритетом данных требований.

В качестве примера предположим, что обслуживается требование j-го приоритета. На вход системы поступает задача i–го приоритета. В случае если i<j, то поступившая задача ставится в очередь. Если i>j, то обслуживание требования j-го приоритета прерывается, и соответственно начинается обслуживание задачи i-го приоритета. По окончании исполнения условия i-го приоритета осуществляется дообслуживание задачи j-го приоритета, в случае, если на обслуживание не поступила новая заявка с наиболее высоким приоритетом.

Средняя продолжительность обслуживания подсистемой «Диспетчер-Процессор» потока задач равняется $\tau_{Д} + \nu_{П}$. Требование, которое поступило в момент, когда система является, занята, становится перед ней в очередь, которая ограничена количеством мест k, и ждет обслуживания. Задачи, которые не попали в очередь ожидания Oi, с вероятностью отказа направляются в иные очереди. В соответствии с выражениями (9) и (10) устанавливаются коэффициенты передач α_i и вероятности отказа в обслуживании i-й СМО ($P_{откi}$).

Средний период ожидания задач с абсолютными приоритетами в очереди перед i-й системой «Диспетчер-Процессор» определиться согласно следующему выражению:

$$w_k^{АП} = \frac{\sum_{i=1}^{H} \lambda_i \cdot t_i \cdot (1 + \nu_{t_i}^2)}{2 \cdot (1 - \sum_{i=1}^{k-1} \rho_i) \cdot (1 - \sum_{i=1}^{k} \rho_i)} + \frac{\sum_{i=1}^{k-1} \rho_i \cdot t_i}{1 - \sum_{i=1}^{k-1} \rho_i} \quad (k = 1,..., H), \quad (15)$$

где ρ_i – является коэффициентом загрузки системы ($\rho_i = \frac{\lambda_i}{\mu_i}$, λ_i – является интенсивностью поступления задач i-го приоритета; μ_i – является интенсивность обслуживания задач i-го приоритета), t_i – является среднее время обслуживания задач i-го приоритета ($t_i = \frac{1}{\mu_i}$), ν_{t_i} – является коэффициентом вариации продолжительности обслуживания задачи i-го приоритета.

Далее рассчитаем среднюю продолжительность ожидания абсолютно всех приоритетных задач во всех очередях рассматриваемой системы:

$$W = \sum_{i=1}^{n-1} \alpha_i \cdot \sum_{k=1}^{H} w_k^{АП}, \; i=1, 3, \ldots, n\text{-}3, n\text{-}1. \qquad (16)$$

Период ответа системы с ДИСПЕТЧЕРАМИ ЗАДАЧ и стратегией распределенной очереди задач при неоднородном входящем потоке задач с абсолютными приоритетами составит следующее:

$$U = \sum_{i=1}^{n} \alpha_i \cdot u_i, \qquad (17)$$

где $u_i = W_i^{АП} + \tau_{обслі}$, $W_i^{АП} = \sum_{i=1}^{k} w_i^{АП}$, $\tau_{обслі} = t_{ДЗ} + \nu_{ПУі}$

2.4 Расчет характеристик планировщиков/диспетчеров задач с бесприоритетным обслуживанием и ограничением числа мест в очередях

Аналитическое моделирование.Параметры задач, а кроме того и архитектурные параметры базовой и исследуемой моделей (быстродействие процессора, период кванта, интенсивность входного потока задач, количество процессоров, средняя загрузка процессора и многое другое) являются одинаковыми.Отличием считаются очереди перед процессорными узлами (в основном варианте они приняты неограниченной длины, в варианте с распределенными диспетчером задач со стратегией распределенной очереди – они являются ограниченными, помимо этого отличительной специфической особенностью выступает наличие перераспределения задач согласно очередям с целью обеспечения балансировки загрузки системы).

С целью проведения вычислительного эксперимента возьмем систему с четырьмя ПРОЦЕССОРАМИ, а помимо этого и с четырьмя ДИСПЕТЧЕРАМИ ЗАДАЧ. Представим граф подобной системы (рис.4.6). В данном случае S0 является источником задач: S1, S2, S3, S4, S5, S6, S7, S8 являются подсистемами «Диспетчер-Процессор» (причем, S1, S3, S5, S7 представляют собой процессорные узлы системы, а S2, S4, S6, S8 – представляют собой диспетчеры).

Распределение задач от источника согласно очередям является равновероятным. Вероятность того, что задача будет перераспределена, согласно иным очередям после диспетчирования составит следующее: при длинных задачах составляет 0,05. Вероятность выдачи результата обслуженной задачи пользователю

при длинных задачах составляет порядка 0,05; при средних задачах порядка 0,3; при коротких задачах порядка 0,7. Вероятность дообслуживания длинной задачи составляет порядка 0,9; средней порядка 0,65; короткой порядка 0,25. Отталкиваясь от представленных сведений, рассмотрим МПС с четырьмя ПРОЦЕССОРАМИ.

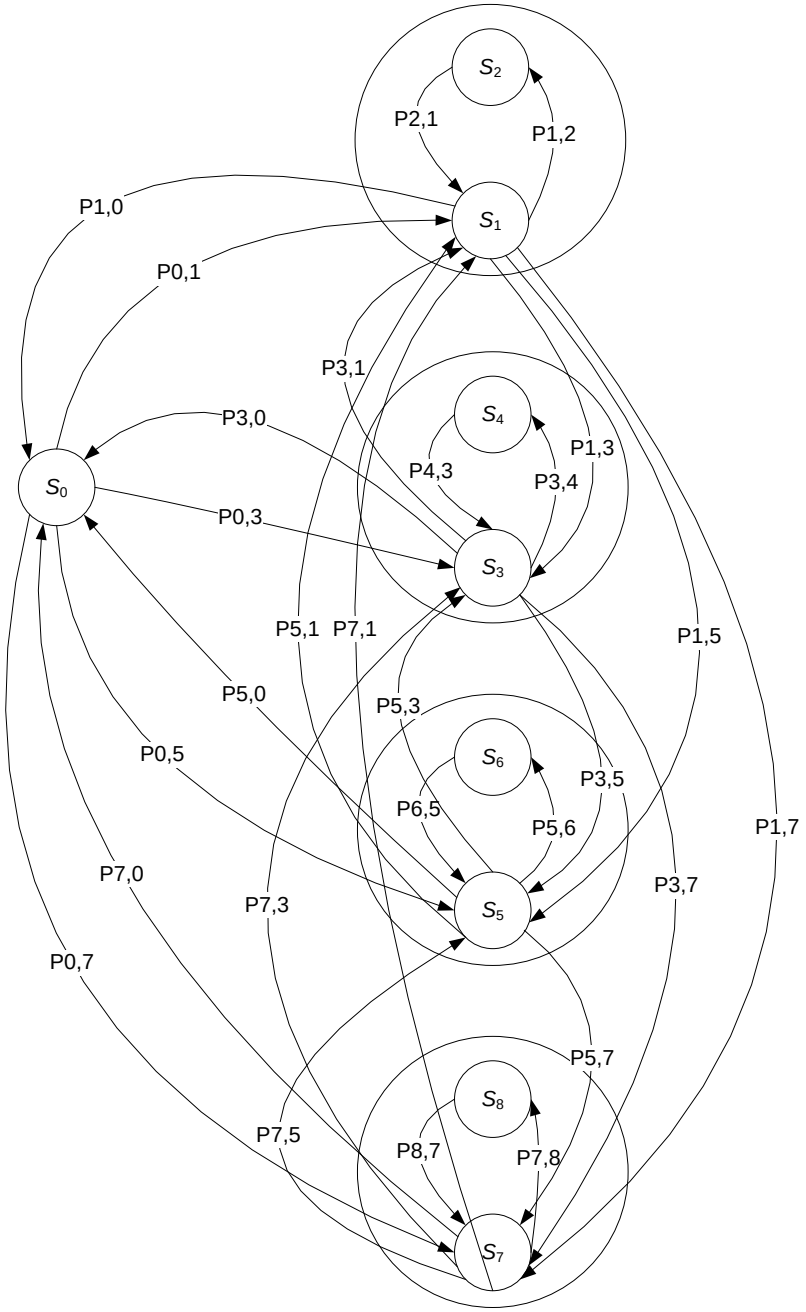


Рисунок 24. Граф системы с 4мя подсистемами «диспетчер задач – процессорный узел»

Система уравнений с целью нахождения интенсивностей потоков задач будет представлена таким образом, как:

$$\begin{cases} \lambda_0 = P_{10}\lambda_1 + P_{30}\lambda_3 + P_{50}\lambda_5 + P_{70}\lambda_7; \\ \lambda_1 = P_{01}\lambda_0 + P_{21}\lambda_2 + P_{31}\lambda_3 + P_{51}\lambda_5 + P_{71}\lambda_7; \\ \lambda_2 = P_{12}\lambda_1; \\ \lambda_3 = P_{03}\lambda_0 + P_{13}\lambda_1 + P_{43}\lambda_4 + P_{53}\lambda_5 + P_{73}\lambda_7; \\ \lambda_4 = P_{34}\lambda_3; \\ \lambda_5 = P_{05}\lambda_0 + P_{15}\lambda_1 + P_{35}\lambda_3 + P_{63}\lambda_6 + P_{75}\lambda_7; \\ \lambda_6 = P_{56}\lambda_5; \\ \lambda_7 = P_{07}\lambda_0 + P_{17}\lambda_1 + P_{37}\lambda_3 + P_{57}\lambda_5 + P_{87}\lambda_8; \\ \lambda_8 = P_{78}\lambda_7. \end{cases} \quad (18)$$

В программе, согласно типу решаемой системой задачи, заполняем матрицу вероятностей передач с целью того, чтобы она смога вычислить значения интенсивностей переходов потоков задач между СМО сети. Помимо этого вычисляются показатели коэффициентов передач между устройствами. К примеру, для системы (4.23) решение при обработке длинных задач будет представлена следующим образом: принимая во внимание равновероятное распределение согласно очередям ($P_{01} = P_{03} = P_{05} = P_{07} = 0,25$), вероятности выдачи результат обработанной длинной задачи ($P_{10} = P_{30} = P_{50} = P_{70} = 0,05$), помимо этого вероятности отправки задачи на дообработку ($P_{12} = P_{34} = P_{56} = P_{78} = 0,9$), кроме того вероятности перераспределения задач согласно иным очередям в процессе переполнения текущей очереди и появления отказа ($P_{13} = P_{15} = P_{31} = P_{35} = P_{51} = P_{57} = P_{73} = P_{75} = 0,017;\ P_{17} = P_{37} = P_{53} = P_{71} = 0,016$).

Назначение условий согласно очередям исследуемой сети выбрано равновероятное для того, чтобы приближено оценить картину математического моделирования фактического процесса, с целью избегания перегрузки системы, когда все задачи будут пытать приобрести обслуживание в одном, либо же нескольких процессоров, а некоторые процессоры вообще будут простаивать. $\lambda_i = \lambda_0 / n$ при равной вероятности распределения запросов (непосредственно, в частном случае).

$$w_i = \sum_{i=1}^{n} \frac{\lambda_i + \sum_{\forall n(n \neq i)} p_{ji} \cdot \lambda_i}{\lambda_0} \cdot \frac{L_i}{\lambda_i}. \qquad (19)$$

Продолжительность ответа в системе с распределенными диспетчерами составляет следующее:

$$U = \sum_{i=1}^{n} \alpha_i \cdot u_i, \qquad (20)$$

где $u_i = w_i + k \cdot (t_k + \delta + \tau + \varsigma)$.

$$U = \sum_{i=1}^{n} \frac{\lambda_i + \sum_{\forall n(n \neq i)} p_{ji} \cdot \lambda_i}{\lambda_0} \cdot \left(w_i + k \cdot (t_k + \delta + \tau + \varsigma)\right). \qquad (21)$$

Результат расчета приведен в таблице 3.

Таблица.3. Результаты расчета.

Среднее время ответа	3,6600	3,6869	3,5000
Среднее время ожидания заявок	1,6650	1,1200	0,6200
Среднее время ожидания(проц)	0,4790	0,0401	0,0208
Среднее время ожидания(дисп)	0,0006	0,0003	0,0001
Среднее время ответа(проц)	0,1730	0,1355	0,1190
Среднее время ответа(дисп)	0,0100	0,0100	0,0100
Ia=1.4			
Загрузка диспетчера задач	0,0630	0,0380	0,0200
Загрузка процеесоров	0,7000	0,4200	0,2400
Время ответа	3,9700	3,9200	3,6200
Время ожидания заявок	1,9700	1,3700	0,7400
Среднее время ожидания(проц)	0,1060	0,0490	0,0580
Среднее время ожидания(дисп)	0,0006	0,0003	0,0002
Среднее время ответа(проц)	0,1940	0,1435	0,0245
Среднее время ответа(дисп)	0,0100	0,0100	0,0100
Ia=1.6			
Загрузка диспетчера задач	0,0720	0,0440	0,0200
Загрузка процеесоров	0,8000	0,4800	0,2700
Время ответа	4,5100	4,1680	3,7300
Время ожидания заявок	2,6300	1,6400	0,8600
Среднее время ожидания(проц)	0,1300	0,0595	0,2850
Среднее время ожидания(дисп)	0,0007	0,0004	0,0002
Среднее время ответа(проц)	0,2150	0,1525	0,1255
Среднее время ответа(дисп)	0,0100	0,0100	0,0100
Ia=1.8			
Загрузка диспетчера задач	0,0810	0,0500	0,0300
Загрузка процеесоров	0,9000	0,5400	0,3000
Время ответа	4,9000	4,4400	3,8600
Время ожидания заявок	3,0900	1,9400	0,9975
Среднее время ожидания(проц)	0,1530	0,0705	0,0320
Среднее время ожидания(дисп)	0,0008	0,0005	0,0002
Среднее время ответа(проц)	0,2350	0,1625	0,1290
Среднее время ответа(дисп)	0,0100	0,0100	0,0100

Результаты экспериментов. В процессе выполненного вычислительного эксперимента поменялось время исполнения задач (малое – для задач, которые требуют высокой реакции на запрос, а что касаемо среднего и высокого времени, то для задач, которые требуют низкой реакции на запрос). Загрузка процессоров прибывала на уровне 65%, что соответствует, непосредственно, средней загрузке системы. Количество процессоров прибывало в диапазоне от 2 до 20. Трудоемкость задач принята такой: для задач, которые требуют высокой реакции – 100мкс, для задач, которые обладают средней трудоемкостью – 500мкс, а помимо этого и для самых трудоемких задач – 1000мкс. Период кванта для проведенных опытов принято регулярным и равным 100мкс. Продолжительность работы диспетчера задач при переключении контекста задач составляет порядка 5мкс; период перезагрузки кэш-памяти принято равным порядка 5мкс.

Из представленной ранее таблицы вытекает то, что загрузка диспетчера задач со стратегией распределенной очереди увеличивается с сокращением требуемого времени исполнения задач (повышением скорости реакции МПС).

Система с относительными приоритетами. Исследуем многопроцессорную систему с ДИСПЕТЧЕРОМ ЗАДАЧ, в которую поступает три приоритетных класса задач на обслуживание. При всем при этом задачи первого приоритета являются сверхсрочными; второго приоритета – срочными; третьего приоритета являются обычными, которые не требуют оперативно реакции и результата. Требования приходят в одну общую очередь перед СМО «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР». В соответствии с полученным выражением (2.24) выведем формулы с целью нахождения среднего периода ожидания задач, трех упомянутых приоритетов. Обозначим следующее: $w_1^{ОП}$ - является временем ожидания в очереди требований первого приоритета, $w_2^{ОП}$ - является временем ожидания в очереди требований второго приоритета, $w_3^{ОП}$ - является временем ожидания в очереди требований третьего приоритета. Коэффициент вариации необходимо принять равным 1.

Представим полученные выражения следующим образом:

$$w_1^{ОП} = \frac{\rho_1 \cdot t_1 + \rho_2 \cdot t_2 + \rho_3 \cdot t_3}{(1-\rho_1)}; \qquad (22)$$

$$w_2^{ОП} = \frac{\rho_1 \cdot t_1 + \rho_2 \cdot t_2 + \rho_3 \cdot t_3}{(1-\rho_1)\cdot(1-\rho_1-\rho_2)}; \qquad (23)$$

$$w_3^{ОП} = \frac{\rho_1 \cdot t_1 + \rho_2 \cdot t_2 + \rho_3 \cdot t_3}{(1-\rho_1-\rho_2)\cdot(1-\rho_1-\rho_2-\rho_3)}, \qquad (24)$$

где ρ_1, ρ_2, ρ_3 – является загрузкой подсистемы «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» задачами 3х приоритетных классов ($\rho_1 = \frac{\lambda_1}{\mu_1}, \rho_2 = \frac{\lambda_2}{\mu_2}, \rho_3 = \frac{\lambda_3}{\mu_3}$), t_1, t_2, t_3 – является средним временем обслуживания задач 1го, 2го и 3го приоритетного класса ($t_1 = \frac{1}{\mu_1}, t_2 = \frac{1}{\mu_2}, t_3 = \frac{1}{\mu_3}$).

Далее произведем расчет среднего времени ожидания приоритетных задач в очередях сети:

$$W = \sum_{i=1}^{n-1} \alpha_i \cdot \sum_{k=1}^{H} w_k^{ОП},\ n = 1,3,...,n-1.$$

$$W = \alpha_1 \cdot (w_1^{ОП} + w_2^{ОП} + w_3^{ОП}) + ... + \alpha_{n-1} \cdot (w_1^{ОП} + w_2^{ОП} + w_3^{ОП}) = \\ = (w_1^{ОП} + w_2^{ОП} + w_3^{ОП}) \cdot (\alpha_1 + \alpha_2 + ... + \alpha_{n-1}) \qquad (25)$$

Период ответа системы с ДИСПЕТЧЕРОМ ЗАДАЧ, а помимо этого и стратегией разделения согласно пространству при неоднородном входящем потоке задач с относительными приоритетами составит следующее:

$$U = \sum_{i=1}^{n} \alpha_i \cdot u_i \qquad (26)$$

где $u_i = W_i^{ОП} + \tau_{обслі}$, $W_i^{ОП} = \sum_{i=1}^{k} w_i^{ОП}$, $\tau_{обслі} = t_{ДЗ} + \nu_{ПУі}$.

Представленные процедуры осуществлены в программе.

Результаты расчетов основным методом, сформированным методом проиллюстрированы на рис.25.

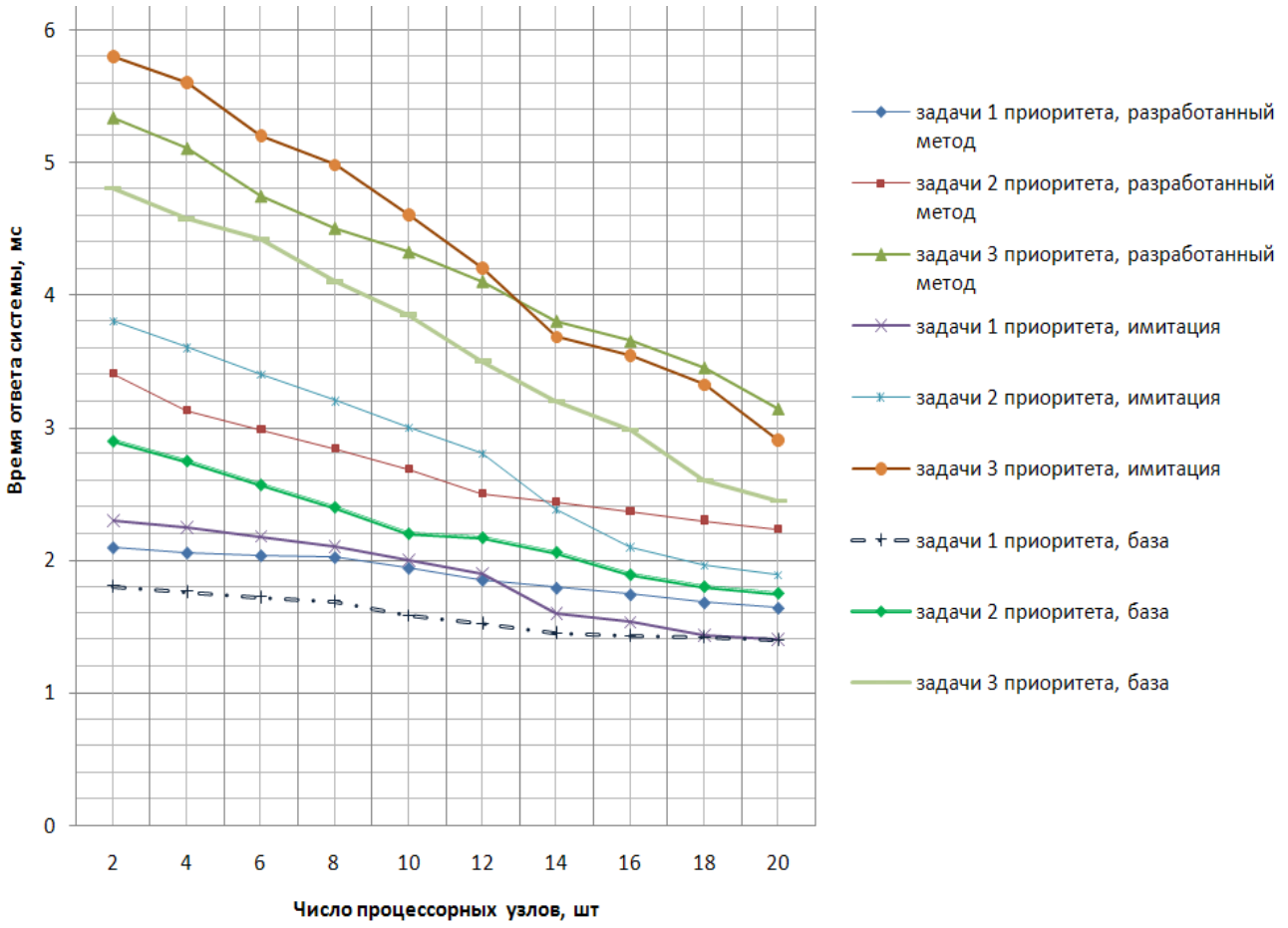


Рисунок 25. Зависимость продолжительности ответа многопроцессорной системы с распределенными диспетчерами задач и относительными приоритетами от количества процессорных узлов

Система с абсолютными приоритетами. Исследуем такую же систему с тремя приоритетными классами задач на обслуживание. В соответствии с полученным выражением (2.27) приведем формулы с целью нахождения среднего времени ожидания задач для трех приоритетов. Обозначим следующее: $w_1^{АП}$ - является временем ожидания в очереди требований первого приоритета, $w_2^{АП}$ - является временем ожидания в очереди требований второго приоритета, $w_3^{АП}$ - является временем ожидания в очереди требований третьего приоритета. Коэффициент вариации необходимо принять равным 1. Далее представим приобретенные выражения.

$$w_1^{АП} = \frac{\rho_1 \cdot t_1}{(1-\rho_1)} \qquad (27)$$

$$w_2^{АП} = \frac{\rho_1 \cdot t_1}{(1-\rho_1)} + \frac{\rho_1 \cdot t_1 + \rho_2 \cdot t_2}{(1-\rho_1)\cdot(1-\rho_1-\rho_2)} \qquad (28)$$

$$w_3^{АП} = \frac{(\rho_1+\rho_2)\cdot t_2}{(1-\rho_1-\rho_2)} + \frac{\rho_1 \cdot t_1 + \rho_2 \cdot t_2 + \rho_3 \cdot t_3}{(1-\rho_1-\rho_2)\cdot(1-\rho_1-\rho_2-\rho_3)} \qquad (29)$$

где ρ_1, ρ_2, ρ_3 – является загрузкой подсистемы «ДИСПЕТЧЕР ЗАДАЧ – ПРОЦЕССОР» задачами 3х приоритетных классов ($\rho_1 = \frac{\lambda_1}{\mu_1}, \rho_2 = \frac{\lambda_2}{\mu_2}, \rho_3 = \frac{\lambda_3}{\mu_3}$), t_1, t_2, t_3 – является средним временем обслуживания задач 1, 2 и 3 приоритетного класса соответственно ($t_1 = \frac{1}{\mu_1}, t_2 = \frac{1}{\mu_2}, t_3 = \frac{1}{\mu_3}$).

Далее требуется рассчитать среднее время ожидания приоритетных задач в очередях сети:

$$\begin{aligned} W &= \alpha_1 \cdot (w_1^{АП} + w_2^{АП} + w_3^{АП}) + ... + \alpha_n \cdot (w_1^{АП} + w_2^{АП} + w_3^{АП}) = \\ &= (w_1^{АП} + w_2^{АП} + w_3^{АП}) \cdot (\alpha_1 + \alpha_2 + ... + \alpha_n) \end{aligned}, \qquad (30)$$

Продолжительность ответа системы с ДИСПЕТЧЕРОМ ЗАДАЧ и стратегией распределенной очереди при неоднородном входящем потоке задач с абсолютными приоритетами составит следующее:

$$U^{АП} = \sum_{i=1}^{n} \alpha_i \cdot u_i \qquad (31)$$

где $u_i = W_i^{АП} + \tau_{обслі}$, $W_i^{АП} = \sum_{i=1}^{k} w_i^{АП}$, $\tau_{обслі} = t_{ДЗ} + \nu_{ПУі}$.

Представленные процедуры осуществлены в программе.

Результаты расчетов базовым методом, сформированы приведены на рисунке 26.

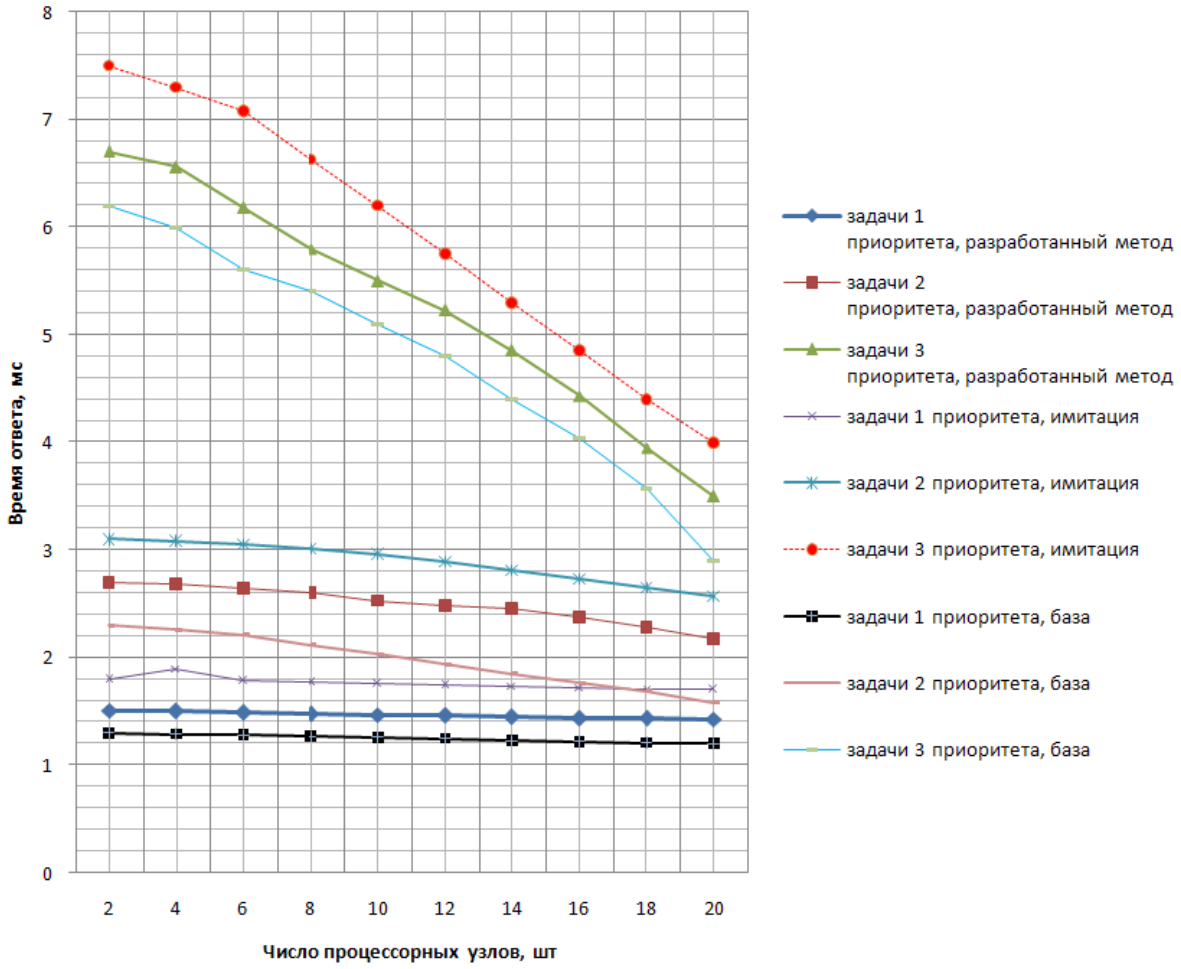


Рис.26. Зависимость времени ответа многопроцессорной системы с распределенными диспетчерами задач и абсолютными приоритетами от количества процессорных узлов

Выводы по разделу 2

Таким образом, основываясь на материале, который изложен в представленном разделе, можно сформулировать следующие выводы:

1. Проведен анализ методов диспетчеризации задач современных операционных систем. Установлено, что в операционных системах общего назначения используются ровно как бесприоритетные, так и приоритетные планировщики, диспетчеры задач. В системах реального времени применяют в большинстве, приоритетные планировщики/диспетчеры.

2. Разработана аналитическая модель планировщиков/диспетчеров со стратегией разделения во времени, а помимо этого и распределенной очередью задач в составе многопроцессорной системы с применением разомкнутых сетей массового обслуживания, которые основаны на бесприоритетных и приоритетных

системах массового обслуживания , а кроме того системах с ограничением длины очереди. Получены выражения для расчета характеристик времени ожидания в очередях, времени ответа системы, вероятностей переполнения очередей.

3. Проведены экспериментальные исследования на полученных моделях. Загрузка диспетчера задач со стратегией распределенной очереди растет с уменьшением времени выполнения прикладных задач (увеличением скорости реакции МПС), что приводит к росту длины очереди и снижению реакции многопроцессорной системы на запрос. Ограничение длины очереди задач уменьшает и стабилизирует время выполнения приоритетных приложений, однако приводит к задержке менее приоритетных.

3. Разработка программы расчета характеристик стохастических сетей массового обслуживания

3.1. Алгоритм расчета характеристик систем и сетей массового обслуживания

Программа предназначена для расчета характеристик стохастических сетей, представляющих модели вычислительных систем, в том числе и многопроцессорных, или их составных частей (отдельных устройств или подсистем). Стохастическая сеть описывается совокупностью параметров, являющихся исходными данными при проведении расчетов. Конфигурация сети задается матрицей вероятностей.

Поскольку разрабатываемая программа кроме анализа характеристик вычислительных систем выполняет также ввод исходных данных и вывод результата, то можно разбить задачу на частные:

- ввод, просмотр и редактирование исходных данных;
- расчет характеристик стохастических сетей массового обслуживания;
- вывод результатов расчетов на экран и в текстовый файл отчета.

Для реализации выделенных задач необходимо разработать отдельные программные модули, причем некоторые функции, выполняемые в этих модулях, также можно выделить в отдельные модули, что значительно упростит процесс

проектирования для разработчика, а также упростит поиск и устранение ошибок, возникающих при создании программы.

К информации, необходимой для проведения расчетов относятся следующие параметры:

- Наименование СМО.
- Время обслуживания.
- Ограничение на длину очереди.
- Система с относительным приоритетом.
- Число классов приоритетов.
- Доля входного потока.
- Количество систем массового обслуживания (СМО) в сети.
- Количество каналов в каждой из СМО.
- Быстродействие каждого канала СМО.
- Матрица вероятностей передач.
- Интенсивность входного потока.

К выходным данным можно отнести результаты расчета, т.е. следующие характеристики СМО:

а) характеристики отдельных СМО:

- Коэффициент передачи.
- Интенсивность входного потока.
- Коэффициент загрузки.
- Среднее число занятых каналов.
- Среднее число заявок в СМО.
- Средняя длина очереди.
- Среднее время ожидания заявки в очереди.
- Среднее время пребывания заявки в очереди.

б) характеристика всей сети в целом:

- среднее количество заявок в сети;
- среднее время пребывания одной заявки в сети;
- среднее время ожидания одной заявки в сети;
- общая длина очередей.

Характеристики отдельных СМО и всей сети в целом будут вычисляться перед сохранением в файл отчета по алгоритмам приведенным ниже. Для разработки алгоритма расчета характеристик стохастических сетей, необходимо ввести некоторые обозначения и описать рассчитываемые характеристики при помощи математических формул.

Для этого рассмотрим произвольную сеть массового обслуживания и введем некоторые обозначения:

- количество СМО в сети H;
- интенсивность входного потока λo;
- количество каналов m1,... ,mн входящих в СМО S1,... ,Sн;
- быстродействие V1,...Vнодного канала СМО S1,... ,Sн;
- средняя трудоемкость обслуживания Q1,...,Qнодним каналом СМО S1,...,Sн.

Матрица вероятностей передач будет иметь следующий вид:

$$p = [p_{ij}] = \begin{array}{c} \\ S_0 \\ \cdot \\ \cdot \\ \cdot \\ S_h \end{array} \begin{array}{c} \begin{array}{cccc} S_0 & \cdot & \cdot & \cdot & S_h \end{array} \\ \begin{bmatrix} P_{00} & \cdot & \cdot & \cdot & P_{0h} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ P_{h0} & \cdot & \cdot & \cdot & P_{hh} \end{bmatrix} \end{array}$$

где So - фиктивная СМО моделирующая источник заявок поступающих с интенсивностью λ0. Для матрицы вероятностей передач должно выполняться условие:

$$\sum_{j=0}^{H} P_{ij} = 1$$

т.е. сумма каждой строки матрицы должна быть равна единице. Это условие вытекает из того, что заявка, выходящая из S1, обязательно попадает в СМО Sj.

Расчет характеристик базируется на результатах и проводится известными методами теории массового обслуживания. Если характеристики сети не меняются

во времени (т.е. Когда средние значения имеют постоянные величины), то в сети существует стационарный режим. Он определяется загрузкой Р и определяется по формуле 32

$$\rho_i = \frac{\lambda_i \upsilon_i}{k_i} < 1 \qquad (32)$$

где λi - интенсивность заявок в СМО;

νi - время обработки заявки в СМО;

ki - количество каналов в СМО.

Таким образом стационарный режим означает, что в отдельных СМО и в сети в целом средние задержки и длины очередей конечны.

Расчет характеристик имеет следующую последовательность:

1. интенсивность потока заявок в СМО Si определяется из формулы:

$$\lambda_i = \sum_{j=0}^{H} \lambda_j \bullet P_{ij} \qquad (i = 1,\ldots,H) \qquad (33)$$

где *Poo* = 0

2. коэффициенты передач для каждой СМО вычисляется по формуле:

$$\alpha_1 = \lambda_i / \lambda_0 \qquad (34)$$

3. определяется среднее время обслуживания νi заявки в СМО S1:

$$\nu = Qi/Vi \qquad (i=1, \ldots, H)$$

$$\rho_0 = \frac{\lambda_i \upsilon_i}{mi} = \frac{\beta_i}{mi} \qquad (35)$$

4. коэффициенты загрузки отдельных СМО вычисляются по формуле: Если в результате расчета выполняется условие существования стационарного режима, т.е. Pi<l (i=l,...,H), можно выполнить расчет следующих характеристик сети.

5.среднее число занятых каналов (среднее число заявок находящихся на обслуживании) определяется по формуле:

$$Ki = mipi \qquad (36)$$

6. средняя длина очереди заявок определяется выражением:

$$l_i = \frac{m_m^{mi} \rho_i^{mi+1}}{m_i(1-\rho_i)^2} \pi_{0i} \qquad (37)$$

где oi- вероятность отсутствия заявок в i-ой СМО, которая определяется по формуле:

7. среднее число заявок в СМО

$$ri=li + k, \qquad (38)$$

8. единичное среднее время ожидания заявок в очереди

$$\pi_{0i} = \left[\frac{(m_i \rho_i)^{mi}}{m_i!(1-\rho_i)} + \sum_{r=0}^{mi-1} \frac{(m_i \rho_i)^r}{r!} \right]^{-1} \qquad (39)$$

$$\omega_i = \frac{l_i}{\lambda_i} = \frac{\upsilon_i m_i \rho_i^{mi}}{m_i!(1-\rho_i)^2} \pi_{0i} \qquad (40)$$

суммарное среднее время ожидания равно:

$$Wi=aiWi \qquad (41)$$

9. единичное среднее время пребывания заявок в СМО представляет собой сумму единичного среднего времени ожидания и среднего времени обслуживания заявки в СМО

$$u_i = w_i + v = \frac{r_i}{\lambda_i} = v \left[\frac{m_i^{mi-1} \rho_i^{mi}}{m_i!(1-\rho_i)^2} \pi_{0i} + 1 \right] \qquad (42)$$

суммарное среднее время пребывания заявки в СМО:

$$Ui=\alpha iui \qquad (43)$$

Характеристики всей сети в целом определяются по следующим формулам:

1. среднее число заявок ожидающих обслуживания в сети, (т.е. общая длина очередей) определяется как сумма средних длин очередей для СМО

2. среднее число заявок, пребывающих в сети вычисляетсяпо формуле:

$$M = \sum_{i=1}^{N} r_i \qquad (44)$$

4. среднее время ожидания заявок в сети равно:

$$W = \sum_{i=1}^{H} W_i \qquad (45)$$

5. среднее время пребывания заявок в сети равно:

$$U = \sum_{i=1}^{H} U_i \qquad (46)$$

На основе приведенных формул составляются алгоритмы, пишутся соответствующие процедуры и функции, с помощью которых производится расчет характеристик сетей.

3.2. Алгоритм расчета характеристик многоканальных СМО с приоритетным обслуживанием

В большинстве моделируемых компьютерных систем требуется расчет многоканальных СМО с приоритетной очередью. К ним относятся модели диспетчеризации задач с разделением времени в многопроцессорных системах [20] как с глобальной, так и с локальными очередями.

Введем допущение, что рассматриваемая система с n каналами является системой с приостановкой (блокированием) заявок (запросов) из i-го приоритетного класса, когда все каналы заняты обслуживанием. На вход СМО поступает суммарный поток заявок от источника с интенсивностями $\lambda_1,...,\lambda_g$, каждый из которых имеет различный приоритет k_i (i=1,…, g). Обозначим нумерацию потоков запросов на исполнение таким образом, что наиболее приоритетный будет под номером 1, следующий по приоритету – 2, наименее приоритетный – g. Все входящие потоки заявок являются пуассоновскими, со временем обслуживания задач распределенными экспоненциально, причем интенсивность входного потока задач k-го потока равна λ_k, а интенсивность обслуживания в процессоре μ_k, k=1, 2, 3, …, g. В таком случае задачи k-го потока обслуживаются лишь в том случае, когда отсутствуют задачи более высокоприоритетных потоков с номерами 1, 2, …, k-1. Аналитический анализ эффективности такой системы с общей очередью для частного случая суммарного

потока, состоящего из двух потоков, становится очень трудоемкой задачей, требующий громоздких выкладок и преобразований. Однако численный метод оценки производительности можно получить с использованием некоторых соображений [11].

Вероятность приостановки задачи первого приоритета равна вероятности того, что все каналы заняты обслуживанием ранее поступивших заявок первого приоритета:

$$P_{s_1} = \frac{\omega_1^n}{n! \cdot \sum_{i=0}^{n} \frac{\omega_1^i}{i!}} \qquad (47)$$

где $\omega_1 = \lambda_1 / \mu_1$ - загрузка каналов заявками первого приоритета; λ_1 – интенсивность потока заявок первого приоритета; μ_1 – интенсивность обслуживания потока заявок первого приоритета.

Среднее число каналов, занятых обслуживанием заявок первого приоритета, рассчитывается по формуле [12]:

$$\overline{C_1} = \omega_1 \cdot \left(1 - P_{s_1}\right) \qquad (48)$$

В таком случае для обслуживания второго и последующих по приоритетности потоков заявок свободны в среднем $n - \overline{C_1}$ каналов и, ввиду того, что задачи второго типа имеют абсолютный приоритет перед всеми остальными, то эффективность их обслуживания также будет описываться формулами Эрланга.

Вероятность приостановки задачи второго приоритета равна:

$$P_{s_2} = \frac{\omega_2^{n-\overline{C_1}}}{\left(n - \overline{C_1}\right)! \cdot \sum_{i=0}^{n-\overline{C_1}} \frac{\omega_2^i}{i!}}, \qquad (49)$$

где $\omega_2 = \lambda_2 / \mu_2$ - загрузка каналов задачами второго приоритета; λ_2 – интенсивность потока задач второго приоритета; μ_2 – интенсивность обслуживания потока задач второго приоритета.

Среднее число каналов, занятых обслуживанием задач второго приоритета равно:

$$\overline{C_2} = \omega_2 \cdot \left(1 - P_{s_2}\right) \qquad (50)$$

В приведенных соотношениях значение $(n - \overline{C_1})$ означает среднее число каналов, свободных для обслуживания второго и последующих приоритетных потоков заявок.

Для потока *k*-го приоритета выражения, оценивающие эффективность обслуживания заявок данного приоритета, представлены ниже:

$$P_{s_k} = \frac{\omega_k^{\,n - \sum_{j=1}^{k-1} \overline{C_j}}}{\left(n - \sum_{j=1}^{k-1} \overline{C_j}\right)! \cdot \sum_{i=0}^{n - \sum_{j=1}^{k-1} \overline{C_j}} \frac{\omega_k^i}{i!}} = \frac{\omega_k^a}{a! \cdot \sum_{i=0}^{a} \frac{\omega_k^i}{i!}}, \qquad (51)$$

где $a = n - \sum_{j=1}^{k-1} \overline{C_j}$.

Среднее число каналов, занятых обслуживанием заявок *k*-го приоритета, рассчитывается по формуле:

$$\overline{C_k} = \omega_k \cdot \left(1 - P_{s_k}\right) \qquad (52)$$

Если округлять $\overline{C_k}$ до ближайшего целого числа, то вносится некоторая погрешность в результаты расчетов. Она по сравнению с имитационной моделью составляет около 7% и падает с ростом загрузки ω и числа процессоров n. Основная доля погрешности связана с округлением дробных чисел до целых значений, однако её можно существенно сократить использованием гамма-функции при вычислениях факториала. При вычислениях факториалов для больших *n* возможно использование формулы Стирлинга, которая при малой вычислительной трудоемкости дает приближенное значение с достаточно высокой точностью [22].

Пропускная способность заявки, имеющей *k*-й приоритет, определяется по формуле:

$$A_k = (1 - P_{sk}) \cdot \lambda_k , \qquad (53)$$

где P_{sk} – вероятность приостановки обслуживания заявок *k*-го приоритета, λ_k – интенсивность потока заявок *k*-го приоритета.

$$P_{s_k} = \frac{\omega_k^{\,n-\sum_{j=0}^{k-1}\overline{C_j}}}{\left(n-\sum_{j=0}^{k-1}\overline{C_j}\right)! \cdot \sum_{i=0}^{n-\sum_{j=0}^{k-1}\overline{C_j}} \frac{\omega_k^i}{i!}} = \frac{\omega_k^a}{a! \cdot \sum_{i=0}^{a} \frac{\omega_k^i}{i!}}, \quad k = \overline{1,q} \qquad (54)$$

где $\omega_k = \lambda_k / \mu_k$ - загрузка процессоров заявками *k*–го приоритета; λ_k – интенсивность потока задач *k*–го приоритета; μ_k – интенсивность обслуживания потока задач *k*–го приоритета; $\overline{C_{k-1}} = \omega_{k-1} \cdot \left(1 - P_{s_{k-1}}\right)$.

Формула (2.40) позволяет рассчитывать рекуррентно все значения P_{s_k}, при этом j следует изменять в пределах j=0, 1,2…(*k*-1):

Выражение $a = n - \sum_{j=0}^{k-1} \overline{C_j}$ вычисляет среднее число процессоров, свободных для обслуживания потоков задач *k*-го и последующих более низких приоритетных классов, где $\sum_{j=0}^{k-1} \overline{C_j}$ - суммарное число процессоров, занятых обслуживанием задач 1, 2,…, *k*-1 приоритетных классов, причем $C_0 = 0$.

Приведенная модель массового обслуживания была реализована в виде отдельного модуля для расчета характеристик приоритетных планировщиков (диспетчеров) задач с глобальной очередью для многопроцессорных систем.

3.3. Структура входных и выходных данных

Для нормальной работы программной системы необходимо определить формат и структуру входных, выходных и промежуточных данных.Информация будет храниться в файле. Опишем ее в виде таблицы.

Таблица 3. Описание структуры данных.

Назначение	Имя	Тип	Раз
Интенсивность входного потока	Lamda	double	5
Интенсивность обслуживания	Mu	double	5
Число занятых процессоров	omega	double	5
Приоритет	k	double	5
Вероятность приостановки обслуживания	P	double	5
Среднее число процессоров	C	double	5
Пропускная способность	A	double	5

Файл будет иметь количество полей и строк пропорциональное количеству сетей.

$$p = [p_{ij}] = \begin{array}{c|cccc} & S0 & \cdots & S50 \\ S0 & P00 & \cdots & .. \\ \vdots & \vdots & \cdots & \vdots \\ S50 & .. & \cdots & Pmn \end{array}$$

3.4 Разработка алгоритмов решения задачи

Выше была обозначена последовательность действий выполняемых программой для решения поставленной задачи и была определена структура данных. Рассмотрим подробнее алгоритм работы каждого отдельного действия.

После загрузки программы на исполнение открывается окно ввода параметров отдельных СМО. При запуске построенной модели на исполнение проверяются введенные параметры на допустимость, если все они верны, запускается расчет характеристик по заданным параметрам.

Количество СМО добавляется автоматически по желанию пользователя. Далее пользователю необходимо выбрать количество процессоров, количество задач и набор параметров, таких как интенсивность входного потока, интенсивность обслуживания и приоритет. Если введен один набор параметров, то после нажатия на кнопку расчитать программа проводит проверку входных данных. При некорректном вводе происходит формирование вывода ошибки и в зависимости от ошибки будет выдано соответствующее ей сообщение. В случае, когда все введенные данные корректны, программа посчитает характеристики и выдаст полученные данные в поля вывода. При завершении ввода более одного набора параметров СМО нажатием кнопки расчитать программа построит график.

Полученные данные можно сохранить в виде таблицы в текстовый файл.

При необходимости, программа предоставляет возможность сбросить все параметры и введенные данные.

UML-диаграмма алгоритма ввода начальных параметров и проверки их корректности приведена на рисунке 26.

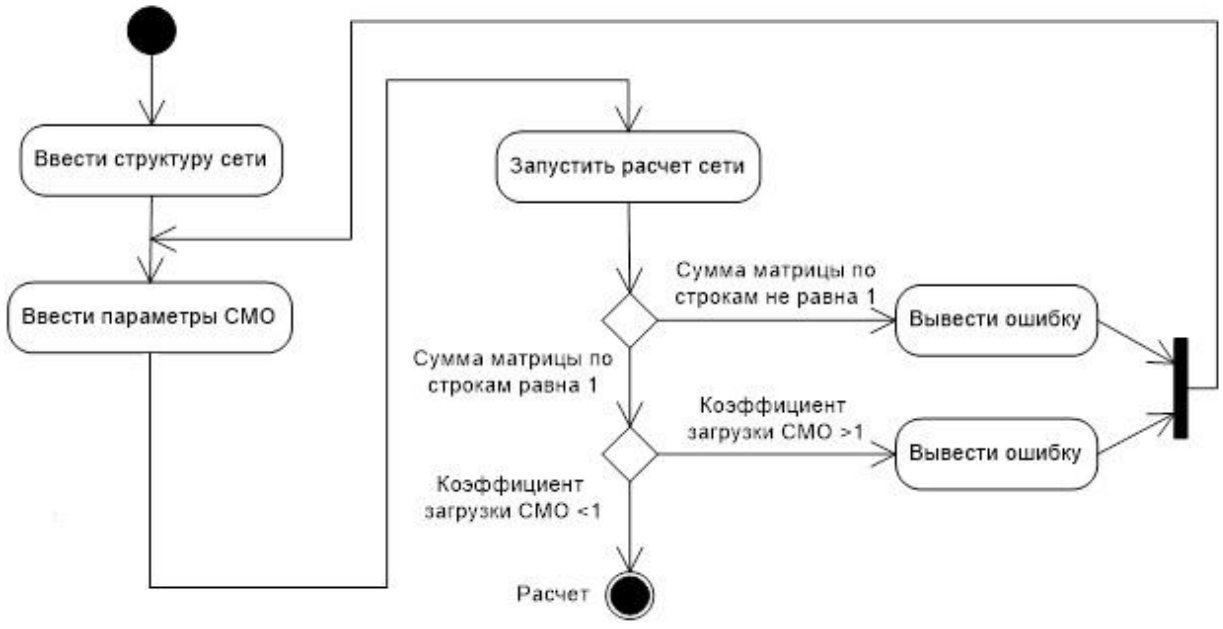


Рисунок 26. Схема алгоритма ввода начальных параметров их проверка

Основным является модуль, в котором производятся расчеты математической модели СМО и отдельных СМО, в основе которых лежат положения ТМО. В общем случае, статистические характеристики СМО складываются из характеристик СМО, входящих в нее, а те, в свою очередь, от

характеристик, рассчитанных для каждого канала, приоритета ряда ограничений внутри данных СМО (ДЗ, ПУ и др.). Расчет проводится согласно оптимизированного под программу теоретико-математическому аппарату теории ВС [16, 67, 107] и представляет собой реализованные на языке программирования выражения и формулы.

В соответствии с технологией RAD (rapidapplicationdevelopment — быстрая разработка приложений) выделено три этапа в разработке программы [9, 34]:

- разработка графического интерфейса для взаимодействия пользователя с программой;
- разработка модулей, решающих задачи, определенные выше;
- привязка реализации модулей к графическому интерфейсу.

Графический интерфейс пользователя – разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.) исполнены в виде графических изображений.

Ниже представлено основное окно программы с панелью выбора параметров, формой для ввода набора входных данных и пустой график:

- основное окно предоставляет пользователю возможность выбрать число процессоров, использовать гамма-функция для расчета факториала, очистить все входные данные и расчитать характеристики системы, исходя из введенных данных;

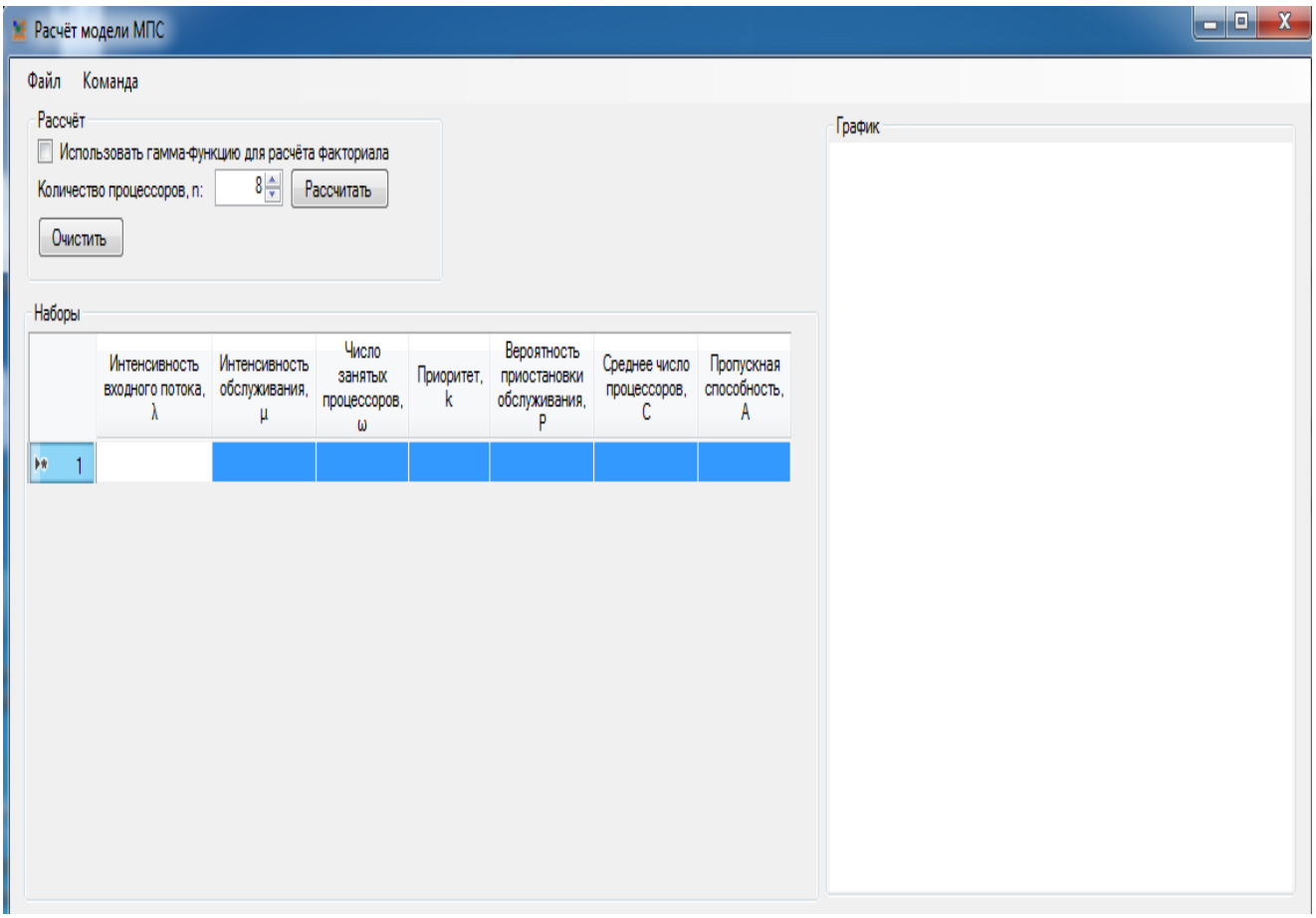


Рисунок 27.Основное окно программы.

– окно файл позволяет пользователю выбрать файл, для загрузки структурированных входных данных, предоставляет возможность сохранить полученные результаты в файл и завершить работу исходной программы;

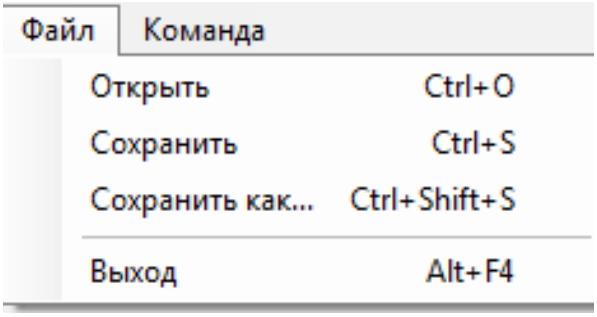


Рисунок 29. Окно параметров СМО

– окно команда предоставляет выбор действия «рассчитать» для расчета характеристик системы, а так же действие «очистить» для удаления введенных ранее данных в форму;

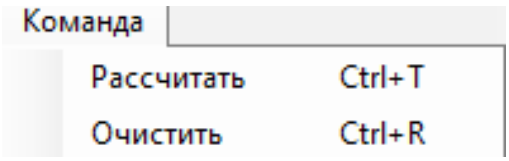
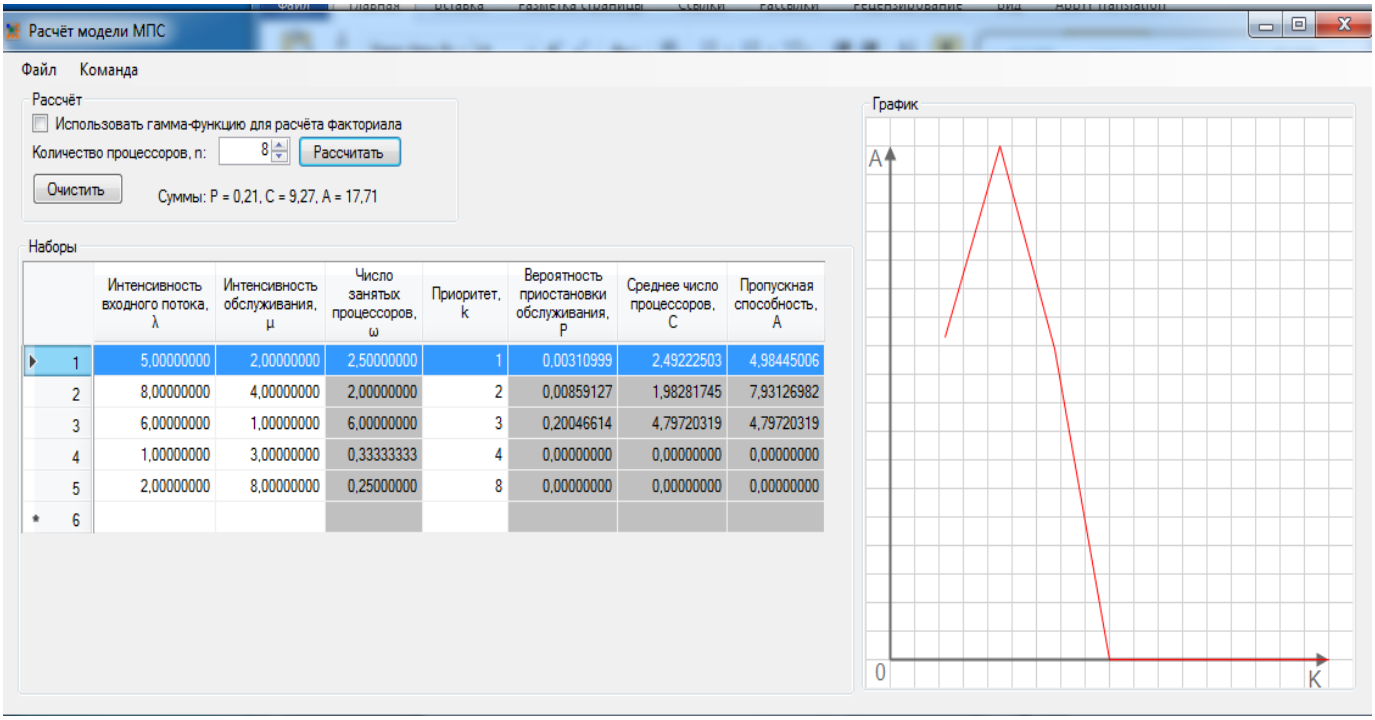


Рисунок 30. Окно команда.

– основное окно после ввода наборов данных и расчета характеристик выводит в окна вывода полученные результаты и на их основе строит график.



	Интенсивность входного потока, λ	Интенсивность обслуживания, μ	Число занятых процессоров, ω	Приоритет, k	Вероятность приостановки обслуживания, P	Среднее число процессоров, C	Пропускная способность, A
1	5,00000000	2,00000000	2,50000000	1	0,00310999	2,49222503	4,98445006
2	8,00000000	4,00000000	2,00000000	2	0,00859127	1,98281745	7,93126982
3	6,00000000	1,00000000	6,00000000	3	0,20046614	4,79720319	4,79720319
4	1,00000000	3,00000000	0,33333333	4	0,00000000	0,00000000	0,00000000
5	2,00000000	8,00000000	0,25000000	8	0,00000000	0,00000000	0,00000000
* 6							

Рисунок 31. Основное окно после ввода данных и расчета характеристик

Рассмотрим типичный порядок действий, которые должен выполнить пользователь. После запуска программы можно сразу приступить к вводу входных данных и расчету характеристик МПС. В процессе создания либо после пользователь может перейти к редактированию входных данных. Можно выполнять определенные действия с помощью меню и кнопочного интерфейса. Кроме того, сразу после загрузки программы пользователь может открыть ранее созданный проект нажав на иконку папки или через меню «Файл> Открыть». Для сохранения проекта можно использовать кнопку с изображением дискеты или через меню «Файл> Сохранить».

После заполнения параметров МПС, для дальнейшей работы необходимо отправить программу на моделирование, нажав на кнопку расчитать или через меню «Команда> Рассчитать», после чего пользователю будет предоставлена

возможность использовать расчеты с варьируемыми параметрами и задавать эти параметры. Далее необходимо нажать кнопку «Рассчитать». После проведения необходимых расчетов программа выведет в поля вывода полученные результаты, имеется возможность проанализировать полученные статистические данные.

При отправке комплекса программ на проведения расчета МПС могут выводиться некоторые ошибки, указывающие на неправильные входные параметры (рис. 31):

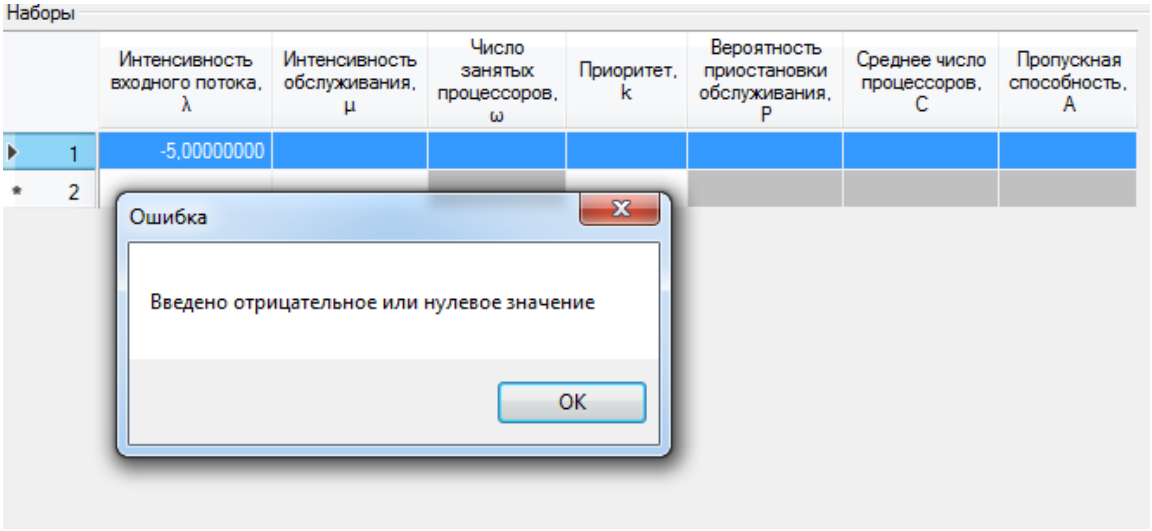


Рисунок 31. Разновидности выводимых ошибок

После обнаружения подобных ошибок требуется вернуться к редактированию входных данных и исправить те из них, не удовлетворяющие условиям.

3.5 Описание программы

В данном подразделе содержится следующая информация:

- общие сведения;
- функциональное назначение;
- описание структуры программы;
- используемые технические средства;
- вызов и загрузка;
- входные данные;
- выходные данные.

Разработанная программа носит название «Программа для расчета стохастических сетей». Программа предназначена для работы в ОС Windows.

Программа предназначена для осуществления следующих функций:

- ввод, просмотр и редактирование исходных данных;
- расчет характеристик многопроцессорных систем;
- вывод результатов расчета на экран или в файл;

Для решения поставленных задач был использован набор функций языка C#.

При проектировании программной системы использовался модульный принцип написания программ, что позволило значительно облегчить разработку программы и ее отладку.

Состав программной системы выглядит следующим образом:

Проект состоит из7классов. Проект скомпилирован в .exeфайлпод названием МПС.exe.

Классы выполняют следующие функции:

1) **Programm**.

Содержит статический метод Main, который включает визуальные стили, устанавливает совместимое отображение текста и запускает основную форму программы.

2) **MpsForm**.

Создает форму для ввода входных данных, вызывает классы для расчета характеристик, создает событие нажатия кнопки расчитать, вызывает класс для отрисовки графика. Класс содержит в себе метод MpsFormFormClosing, который проверяет намерение закрыть приложение, метод ButtonCalculateClick вызывает метод для расчета характеристик МПС, метод ButtonClearClick очищает заполненные ранее данные, метод ShowErrorAndSetOldCellValue отображает ошибки и результаты, полученные при расчете характеристик МПС, метод DrawGraphic строит график по полученным из расчета данным.

3) **Calulator**.

Считывает данные из полей ввода и производит расчет характеристик МПС. Содержит метод GetFactorial для расчета факториала с использованием гамма-

функции и без неё. Метод Gamma рассчитывает факториал с помощью гамма-функции.

4) **XmlFileManager**.

Класс используется для сериализации и десериализации в файл. Содержит методы: Save для записи в файловый поток сериализованного объекта; Opendля получения данных из файлового потока и десериализации объекта.

5) **FileModel**.

Класс предназначен для работы с файлом.

6) **GraphicPoint**.

Данный класс используется для построения графика

7) **Row**.

Создает таблицу для ввода набора данных и поля для вывода результатов расчета характеристик МПС.

Программа вызывается двойным нажатием левой кнопки мыши на пиктограмму программы.

Входными данными для работы программы является информация, необходимая для проведения расчетов (интенсивность входного потока, количество процессоров, интенсивность обслуживания, приоритет);

Выходными данными являются следующие характеристики СМО:

- число заняты процессоров;
- вероятность приостановки обслуживания;
- среднее число процессоров;
- пропускная способность.

После того как программное обеспечение было разработано, появилась необходимость в отладке всех составляющих ее частей. Один из наиболее важных и трудоемких аспектов программирования - это отладка программы. Даже при высокой квалификации программиста неизбежно возникновение ошибок при первых запусках программы. При программировании на языке C# возникновение синтаксических ошибок минимально. Но появление логических ошибок неизбежно. В этом случае необходимо проанализировать программу, выяснить по

какой причине возникли ошибки и каким образом их можно исправить. Проектируемая программа была многократно отлажена на контрольных примерах.

После того как программа была отлажена, возникла необходимость в анализе результатов решения задачи. В ходе проектирования были достигнуты все поставленные перед разработчиком цели. Результаты, полученные в процессе, работы полностью отвечают требованиям, которые предъявлялись в задании на разработку. Кроме того, программное обеспечение гарантирует быстроту доступа к информации, удобный пользовательский интерфейс.

Текст программы приведен в приложении А.

Заключение

Разработана программа для расчета вероятностно-временных характеристик стохастических сетей массового обслуживания и предназначенная для оценки производительности функций ОС. Программа обеспечивает возможность расчета сетей массового обслуживания, содержащих системы массового обслуживания с ограниченной длиной очереди, систем массового обслуживания (как одноканальных, так и многоканальных) с приоритетами, а также расчет сети по нескольким одновременно варьируемым исходным параметрам.

Разработана программа для расчета характеристик стохастических сетей массового обслуживания (например, многопроцессорных систем с диспетчерами задач и другими ресурсами), кроме анализа характеристик многопроцессорных систем выполняет ввод исходных данных и вывод результата. Созданные в программе модели вычислительных систем можно сохранять на диск и загружать сохраненные ранее проекты в виде файлов специализированного формата.

Список литературы

1. ВашкевичН.П., Бикташев Р.А.Недетерминированные автоматыи их использование для реализации систем параллельной обработки информации. (Монография) ,Изд-во ПГУ, 2016-394 с

2. Бикташев Р.А. , ВашкевичН.П.Схемная реализация ядра многопроцессорной операционной системы.Сборник статей Международной научно-технической и научно-методической конференции «Современные технологии в науке и образовании» СТНО-2016 в 4 т. Т.1.- Рязань: Рязан. гос. радиотехн. ун-т, 2016.- с.299-303.

3. ВашкевичН.П., Бикташев Р.А., Киселев С.В. Разработка устройства планирования процессов для многопроцессорных систем. Сборник статей Международной научно-технической и научно-методической конференции «Современные технологии в науке и образовании» СТНО-2016 в 4 т. Т.1.- Рязань: Рязан. гос. радиотехн. ун-т, 2016.- с.304- 308.

4. Бикташев Р.А. Структурная организация многопроцессорных систем с аппаратной поддержкой ядра ОС. Новые информационные технологии и системы: сборник статей XII Международной научно-технической конференции. - Пенза: Изд-во ПГУ, 2015. - С.90-93.

5. Бикташев Р.А. Архитектура аппаратного ядра многопроцессорной операционной системы. Новые информационные технологии и системы: сборник статей XII Международной научно-технической конференции.- Пенза: Изд-во ПГУ, 2015.
- С.86-90.

5. Бикташев Р.А., Вашкевич Н.П.Разработка аппаратно- ориентированного ядра многопроцессорных операционных систем реального времени. Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации. Распознавание –2015: Сборник статей XI международной научно-технической конференции. – Курск: ЮЗГТУ. – 2015. – С. 45 – 47.

6. Вашкевич Н.П., Бикташев Р.А. Формализация алгоритма синхронизации процессов при диспетчеризации задач в многопроцессорных системах с использованием механизма рандеву / Информационные технологии, 2009, № 12,с.12-17.

7. Вашкевич Н.П., Бикташев Р.А. Достоинства формального языка, основанного на концепции недетерминизма, для структурной реализации параллельных систем логического управления процессами и ресурсами / Известия вузов. Поволжский регион. Технические науки. № 1, 2011, с.3-14.

8. Вашкевич Н.П., Бикташев Р.А., Меркурьев А.И. Аппаратная поддержка диспетчера задач с глобальной очередью в многопроцессорных системах// Известия высших учебных заведений. Поволжский регион. Технические науки. № 3, 2011, с.3-12.

9. Волчихин В.И., Вашкевич Н.П., Бикташев Р.А. Планировщик задач с аппаратной поддержкой для многопроцессорных систем// Известия высших учебных заведений. Поволжский регион. Технические науки. № 1, 2012, с.12-21.

10. Вашкевич Н.П., Бикташев Р.А. Формализация алгоритмов управления параллельными процессами на основе использования механизма «рандеву»// «XXI век: Итоги прошлого и проблемы настоящего». Технические науки. Информатика, IT технологии. № 3, 2011, с.111-115.

11.Бикташев Р.А.,Мартышкин А.И. Математическое моделирование диспетчеров задач для систем параллельной обработки на основе разомкнутых систем массового обслуживания. Вмиренаучныхоткрытий. – 2013. – № 6.1 (42) (Математика.Механика. Информатика). – С. 81–101

12. Biktashev R.A., A.I. Martyshkin The mathematical modeling of task managers for multiprocessing systems based on open queueing systems. The collection includes 4th International Conference on Science and Technology, Held by SCIEURO in London, 25-26 June 2014, p.29-33.

13. ВашкевичН.П., Волчихин В.И., Бикташев Р.А. Планировщик задач с аппаратной поддержкой для многопроцессорных системИзвестия высших учебных заведений. Поволжский регион. Технические науки. – 2012. – № 1. – С. 3–14.

14. Ming Liu, Zhonghai Lu, Wolfgang Kuehn, Axel Jantsch. Inter-Process Communication using Pipes in FPGA-based Adaptive Computing / ISVLSI'10, Kefalonia, Greece, 2010 г.

15. БикташевР. А. Комплекс программ для измерения производительности функций операционных систем / Р. А. Бикташев, А. И. Мартышкин // XXI век: Итоги прошлого и проблемы настоящего плюс. Технические науки. Информатика, IT технологии: Научно-методический журнал. – Пенза: Изд-во Пенз. гос. технол. ун-та, №3, 2013 с. 47-54.

16. T. P. Baker, “A comparison of global and partitioned EDF schedulability tests for multiprocessors,” Florida State University, Tech. Rep. TR-051101, 2005.

17. L. Zong, “Nanoprocessors: Configurable hardware accelerators for embedded systems”, Master’s Thesis, 2003

18. Marisabel Guevara, Christopher Gregg.Measuring Basic Linux Operations. CS 656 Operating Systems – Spring 2009.

19. Дейтел Х.М.Операционные системы. Часть1. Основы и принципы.

Х. М. Дейтел, П. Дж. Дейтел, Д. Р. Чофнес**М.:** Бином-Пресс, 2011. -1024 с.

20. Таненбаум Э. Современные операционные системы. 3-е изд. — СПб.: Питер, 2010. — 1120 с.

21. Бикташев Р. А. Математическое моделирование диспетчеров задач со стратегией разделения пространства для параллельных вычислительных систем на основе разомкнутых сетей массового обслуживания/ Р. А. Бикташев, А. И. Мартышкин, //Т 38 Технические науки — от теории к практике. № 9 (22): сборник статей по материалам XXVI международной научно-практической конференции. — Новосибирск: Изд. «СибАК», 2013. — С. 36-42

22. Клейнрок Л. Теория массового обслуживания. – М.: Машиностроение, 1979.-432с.

ПриложениеА

ТекстклассаRun

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Windows.Forms;
using Mps.Enums;
using Mps.Models;
using Mps.Properties;
using Mps.Services;
namespace Mps.Forms
{
   /// <summary>
   /// Основная форма
   /// </summary>
   public partial class MpsForm : Form
   {
      /// <summary>
      /// Предыдущее значение ячейки перед редактированием
      /// </summary>
      private object _cellOldValue;
      /// <summary>
      /// Название файла
      /// </summary>
      private string _fileName;
      /// <summary>
      /// Активирован ли секретный режим
      /// </summary>
      private bool _secretActivated = false;
      /// <summary>
      /// Количество подряд нажатых раз клавиш Shift
      /// </summary>
      private int _shiftKeysCount = 0;
      /// <summary>
      /// Генератор псевдослуайный чисел
      /// </summary>
      private readonly Random _random = new Random();
      /// <summary>
      /// Конструктор
      /// </summary>
```

```
        public MpsForm()
        {
            try
            {
                // Инициализация компонентов
             InitializeComponent();
                // Установить нумерацию строк
               SetRowNumbers();
                // Установить возможно ли добавление кортежей
                SetAllowToAddRows();
            }
            catch (Exception)
            {
                ShowError(Resources.OcceredUnclownExceptionErrorMessage);
            }
        }
        /// <summary>
        /// Событие закрытия формы
        /// </summary>
        /// <param name="sender">Отправитель события</param>
        /// <param name="e">Аргументы</param>
        private void MpsFormFormClosing(object sender, FormClosingEventArgs e)
        {
            try
            {
                // Показать сообщение с вопросом
var result = MessageBox.Show(Resources.ExitQuestionMessage, Resources.ExitCaption,
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
                // Если нажато "Да", то выйти
                if (result == DialogResult.Yes)
                {
                    return;
                }
                // Иначе отменить выход
                e.Cancel = true;
            }
            catch (Exception)
            {
                ShowError(Resources.OcceredUnclownExceptionErrorMessage);
            }
        }
        /// <summary>
        /// Событие активации формы
```

```
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void MpsFormActivated(object sender, EventArgs e)
{
   // Отрисовать
   Draw();
}
/// <summary>
/// Событие нажатия на кнопку "Генерировать"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void ButtonGenerateTupplesClick(object sender, EventArgs e)
{
   try
   {
      // Установить картежи
      SetTupples();
   }
   catch (Exception)
   {
      ShowError(Resources.OcceredUnclownExceptionErrorMessage);
   }
}
/// <summary>
///  Событие нажатия на кнопку "Рассчитать"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void ButtonCalculateClick(object sender, EventArgs e)
{
   try
   {
      // Рассчитать
      Calculate();
   }
   catch (Exception)
   {
      ShowError(Resources.OcceredUnclownExceptionErrorMessage);
   }
}
/// <summary>
```

```
/// Событие нажатия на кнопку "Очистить"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void ButtonClearClick(object sender, EventArgs e)
{
   try
   {
      // Очистить таблицу
      tupplesTable.Rows.Clear();
   }
   catch (Exception)
   {
      ShowError(Resources.OcceredUnclownExceptionErrorMessage);
   }
}
/// <summary>
/// Событие выбора пункта меню "Файл" - "Открыть"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void OpenFileToolStripMenuItemClick(object sender, EventArgs e)
{
   try
   {
      if (openFileDialog.ShowDialog() == DialogResult.OK)
      {
         // Задать имя файла
         _fileName = openFileDialog.FileName;
         // Открыть
         OpenFile();
      }
   }
   catch (Exception)
   {
      ShowError(Resources.OcceredUnclownExceptionErrorMessage);
   }
}
/// <summary>
/// Событие выбора пункта меню "Файл" - "Сохранить"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
```

```
private void SaveFileToolStripMenuItemClick(object sender, EventArgs e)
{
   try
   {
      // Если нет имени файла
      if (string.IsNullOrEmpty(_fileName))
      {
         if (saveFileDialog.ShowDialog() == DialogResult.OK)
         {
            // Задать имя файла
            _fileName = saveFileDialog.FileName;
            // Сохранить
            SaveFile();
         }
      }
      // Иначе
      else
      {
         // Сохранить
         SaveFile();
      }
   }
   catch (Exception)
   {
      ShowError(Resources.OcceredUnclownExceptionErrorMessage);
   }
}
/// <summary>
/// Событие выбора пункта меню "Файл" - "Сохранить как..."
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void SaveFileAsToolStripMenuItemClick(object sender, EventArgs e)
{
   try
   {
      if (saveFileDialog.ShowDialog() == DialogResult.OK)
      {
         // Задать имя файла
         _fileName = saveFileDialog.FileName;
         // Сохранить
         SaveFile();
      }
```

```
    }
    catch (Exception)
    {
        ShowError(Resources.OcceredUnclownExceptionErrorMessage);
    }
}
/// <summary>
/// Событие выбора пункта меню "Файл" - "Выход"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void ExitToolStripMenuItemClick(object sender, EventArgs e)
{
    try
    {
        // Выход
        Close();
    }
    catch (Exception)
    {
        ShowError(Resources.OcceredUnclownExceptionErrorMessage);
    }
}
/// <summary>
/// Событие выбора пункта меню "Команда" - "Сгенерировать"
/// </summary>
/// <param name="sender">Отправитель события</param>
/// <param name="e">Аргументы</param>
private void GenerateToolStripMenuItemClick(object sender, EventArgs e)
{
    try
    {
        // Установить картежи
        SetTupples();
    }
    catch (Exception)
    {
        ShowError(Resources.OcceredUnclownExceptionErrorMessage);
    }
}
/// <summary>
/// Событие выбора пункта меню "Команда" - "Рассчитать"
/// </summary>
```

```
    /// <param name="sender">Отправитель события</param>
    /// <param name="e">Аргументы</param>
    private void CalculateToolStripMenuItemClick(object sender, EventArgs e)
    {
      try
      {
        // Рассчитать
        Calculate();
      }
      catch (Exception)
      {
        ShowError(Resources.OcceredUnclownExceptionErrorMessage);
      }
    }
    /// <summary>
    /// Событие выбора пункта меню "Команда" - "Очистить"
    /// </summary>
    /// <param name="sender">Отправитель события</param>
    /// <param name="e">Аргументы</param>
    private void ClearTableToolStripMenuItemClick(object sender, EventArgs e)
    {
      try
      {
        // Очистить таблицу
        tupplesTable.Rows.Clear();
      }
      catch (Exception)
      {
        ShowError(Resources.OcceredUnclownExceptionErrorMessage);
      }
    }
    /// <summary>
    /// Событие начала редактирования ячейки
    /// </summary>
    /// <param name="sender">Отправитель события</param>
    /// <param name="e">Аргументы</param>
    private void TupplesTableCellBeginEdit(object sender,
DataGridViewCellCancelEventArgs e)
    {
      try
      {
        // Запомнить старое значение ячейки
        var cell = tupplesTable.Rows[e.RowIndex].Cells[e.ColumnIndex];
```

```
            _cellOldValue = cell.Value;
        }
        catch (Exception)
        {
            ShowError(Resources.OcceredUnclownExceptionErrorMessage);
        }
    }
    /// <summary>
    /// Событие завершения редактирования содержимого ячейки
    /// </summary>
    /// <param name="sender">Отправитель события</param>
    /// <param name="e">Аргументы</param>
    private void TupplesTableCellEndEdit(object sender, DataGridViewCellEventArgs e)
    {
        try
        {
            // Текущая ячейка
            var cell = tupplesTable.Rows[e.RowIndex].Cells[e.ColumnIndex];
if (cell?.Value == null)
            {
                return;
            }
            // Значение текущей ячейки
            var stringValue = cell.Value.ToString();
            // Для колонки
            switch ((ColumnIndex)e.ColumnIndex)
            {
                // Если колонка интенсивностей
                case ColumnIndex.LamdaColumnIndex:
                case ColumnIndex.MuColumnIndex:
                    // Попытаться получить значение
                    double outDoubleValue;
                    if (double.TryParse(stringValue, out outDoubleValue))
                    {
                        // Установить значение
  cell.Value = outDoubleValue;
                        // Отменить ввод отрицательного значения
                        if (outDoubleValue <= 0)
                        {
                            // Показать сообщение об ошибке и установить предыдущее
значение ячейки
                            ShowErrorAndSetOldCellValue(cell,
Resources.NegativeOrZeroValueEnteredErrorMessage);
```

```
                }
                    var lamdaCell = tupplesTable.Rows[e.RowIndex].Cells[(int)
ColumnIndex.LamdaColumnIndex];
                    var muCell = tupplesTable.Rows[e.RowIndex].Cells[(int)
ColumnIndex.MuColumnIndex];
if (lamdaCell?.Value == null || muCell?.Value == null)
                    {
                      return;
                    }
                    // Попытаться получить измененные значения
                    // Если удалось получить
                    double outLamdaDoubleValue;
                    double outMuDoubleValue;
if (double.TryParse(lamdaCell.Value.ToString(), out outLamdaDoubleValue) &&
double.TryParse(muCell.Value.ToString(), out outMuDoubleValue))
                    {
                      // Установить значение
                      var omegaCell =
tupplesTable.Rows[e.RowIndex].Cells[(int)ColumnIndex.OmegaColumnIndex];
                      omegaCell.Value = Calculator.CalculateOmega(outLamdaDoubleValue,
outMuDoubleValue);
                    }
                    // Иначе
                    else
                    {
                      // Показать сообщение об ошибке и установить предыдущее
значение ячейки
ShowErrorAndSetOldCellValue(cell,
Resources.NotCorrectIntensivityValuesToCalculateErrorMessage);
                    }
                  }
                  else
                  {
                    // Показать сообщение об ошибке и установить предыдущее
значение ячейки
                    ShowErrorAndSetOldCellValue(cell,
Resources.NotCorrectCellValueErrorMessage);
                  }
                  break;
                // Если колонка приоритетов
                case ColumnIndex.KColumnIndex:
                  // Попытаться получить значение
                  int outIntValue;
```

```
                if (int.TryParse(stringValue, out outIntValue))
                {
                    var ks = new List<int>();
                    // Получить список приоритетов
                    foreach (DataGridViewRow row in tupplesTable.Rows)
                    {
                        // Пропустить пустые ячейки приоритетов
                        var kCell = row.Cells[(int) ColumnIndex.KColumnIndex];
                        int outTempIntValue;
if (kCell?.Value == null || !int.TryParse(kCell.Value.ToString(), out outTempIntValue))
                        {
                            continue;
                        }
                        // Пропустить текущую строку
                        if (e.RowIndex == row.Index)
                        {
                            continue;
                        }
                        // Добавить в список
                        ks.Add(outTempIntValue);
                    }
                    // Если не было изменений
                    if (_cellOldValue != null && outIntValue ==
int.Parse(_cellOldValue.ToString()))
                    {
                        return;
                    }
                    // Если уже имеется указанный приоритет
                    if (ks.Contains(outIntValue))
                    {
                        // Показать сообщение об ошибке и установить предыдущее
значение ячейки
                        ShowErrorAndSetOldCellValue(cell,
Resources.ExistPriorityValueErrorMessage);
                    }
                    else
                    {
                        // Установить значение
                        cell.Value = outIntValue.ToString("D");
                        // Отменить ввод отрицательного значения
                        if (outIntValue <= 0)
                        {
```

```
                        // Показать сообщение об ошибке и установить предыдущее
значение ячейки
                        ShowErrorAndSetOldCellValue(cell,
Resources.NegativeOrZeroValueEnteredErrorMessage);
                    }
                  }
                }
                else
                {
                  // Показать сообщение об ошибке и установить предыдущее
значение ячейки
                  ShowErrorAndSetOldCellValue(cell,
Resources.NotCorrectCellValueErrorMessage);
                }
                break;
            }
            SetRowNumbers();
          }
          catch (Exception)
          {
            ShowError(Resources.OcceredUnclownExceptionErrorMessage);
          }
        }
        /// <summary>
        /// Событие удаления строки
        /// </summary>
        /// <param name="sender">Отправитель события</param>
        /// <param name="e">Аргументы</param>
        private void TupplesTableUserDeletedRow(object sender,
DataGridViewRowEventArgs e)
        {
          try
          {
            // Установить нумерацию строк
            SetRowNumbers();
            // Установить возможно ли добавление кортежей
            SetAllowToAddRows();
          }
          catch (Exception)
          {
            ShowError(Resources.OcceredUnclownExceptionErrorMessage);
          }
        }
```

```
        /// <summary>
        /// Событие добавления строки
        /// </summary>
        /// <param name="sender">Отправитель события</param>
        /// <param name="e">Аргументы</param>
        private void TupplesTableUserAddedRow(object sender,
DataGridViewRowEventArgs e)
        {
            try
            {
                // Установить нумерацию строк
                SetRowNumbers();
                // Установить возможно ли добавление кортежей
                SetAllowToAddRows();
            }
            catch (Exception)
            {
                ShowError(Resources.OcceredUnclownExceptionErrorMessage);
            }
        }
        /// <summary>
        /// Установить картежи
        /// </summary>
        public void SetTupples()
        {
            // Очистить таблицу
      tupplesTable.Rows.Clear();
            // Очистить приоритеты
            Generator.Clear();
            // Количество кортежей
            var tupplesCount = numericUpDownG.Value;
            // Заполнить приоритеты
            Generator.Fill((int) tupplesCount);
            // Рассортировать приоритеты
            if (checkBoxRandomK.Checked)
            {
                Generator.DeSort();
            }
            // Заполнить таблицу картежей
            for (var i = 0; i < tupplesCount; i++)
            {
                var lamdaCurrent = 0.0;
                var muCurrent = 0.0;
```

```
            // Если отмечена случайная генерация интенсивностей
            if (checkBoxRandomTupples.Checked)
            {
                // Генерировать положительное число (0 < число < 10) и округлить до
двух знаков после запятой
                while (lamdaCurrent <= 0)
                {
                    lamdaCurrent = Math.Round(_random.NextDouble() * _random.Next(10),
2);
                }
                // Генерировать положительное число (0 < число < 10) и округлить до
двух знаков после запятой
                while (muCurrent <= 0)
                {
                    muCurrent = Math.Round(_random.NextDouble() * _random.Next(10), 2);
                }
            }
            // Иначе
            else
            {
                // Интенсивность потока задач растёт последовательно на единицу
        lamdaCurrent = i + 1;
                // Интенсивность обслуживания равна единице
                muCurrent = 1;
            }
            // Вычислить значение загрузки процессора
            var omegaCurrent = Calculator.CalculateOmega(lamdaCurrent, muCurrent);
            // Значение приоритета
            var kCurrent = Generator.GetByIndex(i);
            // Добавить кортеж
            tupplesTable.Rows.Add(lamdaCurrent, muCurrent, omegaCurrent, kCurrent);
        }
        // Установить нумерацию строк
        SetRowNumbers();
        // Установить возможно ли добавление кортежей
        SetAllowToAddRows();
    }
    /// <summary>
    /// Установить нумерацию строк
    /// </summary>
    private void SetRowNumbers()
    {
        // Для каждой строки установить индекс + 1
```

```
            foreach (DataGridViewRow row in tupplesTable.Rows)
            {
                row.HeaderCell.Value = $"{row.Index + 1}";
            }
        }
        /// <summary>
        /// Установить возможно ли добавление кортежей
        /// </summary>
        private void SetAllowToAddRows()
        {
            // Установить возможно ли редактирование на основе количества строк в
таблице
            tupplesTable.AllowUserToAddRows = tupplesTable.RowCount <
numericUpDownG.Maximum;
        }
        /// <summary>
        /// Показать сообщение об ошибке и установить предыдущее значение ячейки
        /// </summary>
        /// <param name="cell">Ячейка</param>
        /// <param name="errorMessage">ТекстОшибки</param>
        private void ShowErrorAndSetOldCellValue(DataGridViewCell cell, string
errorMessage)
        {
            // Показать сообщение об ошибке
       ShowError(errorMessage);
            // Восстановить значение
            cell.Value = _cellOldValue;
        }
        /// <summary>
        /// Показать сообщение об ошибке
        /// </summary>
        /// <param name="errorMessage">ТекстОшибки</param>
        private static void ShowError(string errorMessage)
        {
            // Показать сообщение об ошибке
            MessageBox.Show(errorMessage, Resources.ErrorCaption,
MessageBoxButtons.OK);
        }
        /// <summary>
        /// Получить значения ячеек таблицы
        /// </summary>
        /// <returns>Список строк</returns>
        private List<Row> GetRows()
```

```
    {
        return tupplesTable.Rows.Cast<DataGridViewRow>()
            .Select(dataGridViewRow => dataGridViewRow.Cells)
.Select(cells => new Row
            {
                Lamda = ParseDoubleFromCell(cells[(int)ColumnIndex.LamdaColumnIndex]),
                Mu = ParseDoubleFromCell(cells[(int)ColumnIndex.MuColumnIndex]),
                Omega =
ParseDoubleFromCell(cells[(int)ColumnIndex.OmegaColumnIndex]),
                K = ParseIntFromCell(cells[(int)ColumnIndex.KColumnIndex]),
                P = ParseDoubleFromCell(cells[(int)ColumnIndex.PColumnIndex]),
                C = ParseDoubleFromCell(cells[(int)ColumnIndex.CColumnIndex]),
                A = ParseIntFromCell(cells[(int)ColumnIndex.AColumnIndex])
            })
            .Where(r => r.Lamda > 0 && r.Mu > 0 && r.K > 0)
            .ToList();
    }
    /// <summary>
    /// Установить список строк в таблицу
    /// </summary>
    /// <param name="rows">Строки</param>
    private void SetRows(IEnumerable<Row> rows)
    {
        foreach (var row in rows)
        {
            tupplesTable.Rows.Add(row.Lamda, row.Mu, row.Omega, row.K, row.P, row.C,
row.A);
        }
    }
    /// <summary>
    /// Рассчитать
    /// </summary>
    private void Calculate()
    {
        // Получить список строк
        var rows = GetRows();
        // Использовать ли гамма-функцию для расчёта факториала
        var overGammaFunction = checkBoxOverGammaFunction.Checked;
        // Вычислить
        Calculator.Calculate(rows, (int)numericUpDownN.Value, overGammaFunction);
        // Очистить таблицу
  tupplesTable.Rows.Clear();
        // Установить список строк в таблицу
```

```
            SetRows(rows);

            // Установить нумерацию строк
            SetRowNumbers();
            // Суммы значений
            labelTotal.Text = "Суммы: " +
$"P = {rows.Sum(r => r.P).ToString("N2")}, " +
$"C = {rows.Sum(r => r.C).ToString("N2")}, " +
$"A = {rows.Sum(r => r.A).ToString("N2")}";
            // Отрисовать
            Draw();
        }
        /// <summary>
        /// Получить вещественное значение из ячейки
        /// </summary>
        /// <param name="cell">Ячейка</param>
        /// <returns>Вещественное значение ячейки</returns>
        private static double ParseDoubleFromCell(DataGridViewCell cell)
        {
            double outValue;
double.TryParse(cell?.Value?.ToString(), out outValue);
            return outValue;
        }
        /// <summary>
        /// Получить целочисленное значение из ячейки
        /// </summary>
        /// <param name="cell">Ячейка</param>
        /// <returns>Целочисленное значение ячейки</returns>
        private static int ParseIntFromCell(DataGridViewCell cell)
        {
            int outValue;
int.TryParse(cell?.Value?.ToString(), out outValue);
            return outValue;
        }
        /// <summary>
        /// Сохранить в файл
        /// </summary>
        private void SaveFile()
        {
            // Если есть имя файла
            if (!string.IsNullOrEmpty(_fileName))
            {
                // Модель файла
```

```
            var fileModel = new FileModel
            {
               N = (int)numericUpDownN.Value,
               IsRandomTupples = checkBoxRandomTupples.Checked,
               IsRandomPriority = checkBoxRandomK.Checked,
               UseGammaFunction = checkBoxOverGammaFunction.Checked,
               Rows = GetRows()
   };
            // Сохранить
            XmlFileManager.Save(fileModel, _fileName);
MessageBox.Show(Resources.FileSavedSuccessfullyInformationMessage,
Resources.SaveFileCaption,
MessageBoxButtons.OK, MessageBoxIcon.Information);
         }
      }
      /// <summary>
      /// Открыть файл
      /// </summary>
      private void OpenFile()
      {
         // Если есть имя файла
         if (!string.IsNullOrEmpty(_fileName))
         {
            // Модель файла
            var fileModel = XmlFileManager.Open(_fileName);
            // Установить значения настроек
            numericUpDownN.Value = fileModel.N;
            numericUpDownG.Value = fileModel.Rows.Count;
            checkBoxRandomTupples.Checked = fileModel.IsRandomTupples;
            checkBoxRandomK.Checked = fileModel.IsRandomPriority;
            checkBoxOverGammaFunction.Checked = fileModel.UseGammaFunction;
            checkBoxOverGammaFunction.Checked = fileModel.UseGammaFunction;
            // Очистить таблицу
    tupplesTable.Rows.Clear();
            // Установить список строк в таблицу
     SetRows(fileModel.Rows);
            // Установить нумерацию строк
            SetRowNumbers();
         }
      }
      /// <summary>
      /// Нарисовать
      /// </summary>
```

```
        public void Draw()
        {
            using (var graphics = pictureBox.CreateGraphics())
            {
                graphics.SmoothingMode = SmoothingMode.HighQuality;
                // Очистить
                graphics.Clear(pictureBox.BackColor);
                // Нарисовать сетку
            DrawGrid(graphics);
                // Нарисовать оси ординат и абцисс
          DrawArrows(graphics);
                // Нарисовать график
                DrawGraphic(graphics);
            }
        }
        /// <summary>
        /// Нарисовать график
        /// </summary>
        /// <param name="graphics">Объект Graphics</param>
        private void DrawGraphic(Graphics graphics)
        {
            // Получить шаги сетки
            var gridSteps = GetSteps();
            // Начало и конец сетки
            var a0K0Point = new PointF(gridSteps.Width, pictureBox.Bottom - pictureBox.Top
- gridSteps.Height);
            var aMaxKMaxPoint = new PointF(gridSteps.Width * 20, gridSteps.Height);
            // Точки построения графика
            var graphicPoints = GetPoinsForGraphic();
            if (!graphicPoints.Any())
            {
                return;
            }
            // Самое правое значение по горизонтали
            var maxKValue = graphicPoints.Max(point => point.K);
            if (maxKValue == 0)
            {
                return;
            }
            // Самое верхнее значение по вертикали
            var maxAValue = graphicPoints.Max(point => point.A);
            if (maxAValue == 0)
            {
```

```
            return;
        }

        // Шаги графика
var graphicSteps = new SizeF((float) ((aMaxKMaxPoint.X - a0K0Point.X -
gridSteps.Width) / maxKValue),
            (float) ((a0K0Point.Y - aMaxKMaxPoint.Y) / maxAValue));
        // Точки рисования
        var points = new List<PointF>();
        foreach (var graphicPoint in graphicPoints)
        {
points.Add(new PointF(
                (float) (a0K0Point.X + graphicPoint.K * graphicSteps.Width),
(float) (a0K0Point.Y - graphicPoint.A * graphicSteps.Height)));
        }
        using (var pen = new Pen(Color.Red, 1))
        {
            // Округленные начальная и конечная точка
            pen.StartCap = LineCap.Round;
            pen.EndCap = LineCap.Round;
            // Нарисовать линии
graphics.DrawLines(pen, points
                .Where(point => !float.IsInfinity(point.X) && !float.IsNaN(point.X)
&& !float.IsInfinity(point.Y) && !float.IsNaN(point.Y))
.ToArray());
        }
    }
    /// <summary>
    /// Нарисовать оси ординат и абцисс
    /// </summary>
    /// <param name="graphics">Объект Graphics</param>
    private void DrawArrows(Graphics graphics)
    {
        // Получить шаги сетки
        var steps = GetSteps();
        // Координаты линий и обозначений
        var arrowStart = new PointF(steps.Width, pictureBox.Bottom - pictureBox.Top -
steps.Height);
        var verticalArrowEnd = new PointF(arrowStart.X, steps.Height);
        var horisontalArrowEnd = new PointF(pictureBox.Right - pictureBox.Left -
steps.Width, arrowStart.Y);
        var a0K0Point = new PointF(arrowStart.X - 15, arrowStart.Y);
        var aLabelPoint = new PointF(verticalArrowEnd.X - 20, verticalArrowEnd.Y);
```

```
        var kLabelPoint = new PointF(horisontalArrowEnd.X - 20, horisontalArrowEnd.Y +
5);
        // Шрифт
        var font = new Font("Arial", 12);
        // Сплошная кисть
        var brush = new SolidBrush(Color.DimGray);
        // Кисть
        using (var grayArrowPen = new Pen(Brushes.DimGray, 2))
        {
            // Стрелка в конце
            grayArrowPen.CustomEndCap = new AdjustableArrowCap(5, 5);
            // Нарисовать
            graphics.DrawLines(grayArrowPen, new[] {arrowStart, verticalArrowEnd});
            graphics.DrawLines(grayArrowPen, new[] {arrowStart, horisontalArrowEnd});
            graphics.DrawString("0", font, brush, a0K0Point);
            graphics.DrawString("A", font, brush, aLabelPoint);
            graphics.DrawString("K", font, brush, kLabelPoint);
        }
    }
    /// <summary>
    /// Нарисовать сетку
    /// </summary>
    /// <param name="graphics">Объект Graphics</param>
    private void DrawGrid(Graphics graphics)
    {
        // Получить шаги сетки
        var steps = GetSteps();
        // Список наборов координат
        var tuples = new List<Tuple<PointF, PointF>>();
        // Нарисовать горизонтальные линии сетки
        for (float i = 0; i < pictureBox.Bottom; i += steps.Height)
        {
            // Вертикальные координаты
            var lineStartPoint = new PointF(0, i);
            var lineEndPoint = new PointF(pictureBox.Right, i);
            // Добавить
            tuples.Add(new Tuple<PointF, PointF>(lineStartPoint, lineEndPoint));
        }
        // Нарисовать вертикальные линии сетки
        for (float i = 0; i < pictureBox.Right; i += steps.Width)
        {
            // Горизонтальные координаты
            var lineStartPoint = new PointF(i, 0);
```

```
            var lineEndPoint = new PointF(i, pictureBox.Bottom);
            // Добавить
            tuples.Add(new Tuple<PointF, PointF>(lineStartPoint, lineEndPoint));
        }
        // Кисть
        using (var lightGrayPen = new Pen(Brushes.LightGray, 1))
        {
            foreach (var tuple in tuples)
            {
                // Нарисовать
                graphics.DrawLines(lightGrayPen, new[] {tuple.Item1, tuple.Item2});
            }
        }
    }
/// <summary>
    /// Получить шаги сетки
    /// </summary>
    /// <returns>Размеры шагов сетки</returns>
    private SizeF GetSteps()
    {
return new SizeF(
            (pictureBox.Right - pictureBox.Left) / (float) 20,
            (pictureBox.Bottom - pictureBox.Top) / (float) 20
        );
    }
    /// <summary>
    /// Получить точки для построения графика
    /// </summary>
    /// <returns>Список точек для построения графика</returns>
    private List<GraphicPoint> GetPoinsForGraphic()
    {
        return tupplesTable.Rows.Cast<DataGridViewRow>()
            .Select(dataGridViewRow => dataGridViewRow.Cells)
.Select(cells => new GraphicPoint
            {
                K = ParseDoubleFromCell(cells[(int)ColumnIndex.KColumnIndex]),
                A = ParseDoubleFromCell(cells[(int)ColumnIndex.AColumnIndex])
            })
            .Where(point => point.K > 0)
            .OrderBy(point => point.K)
            .ToList();
}
    /// <summary>
```

```
        /// Событие нажатия клавиш на форме
        /// </summary>
        /// <param name="sender">Отправитель события</param>
        /// <param name="e">Аргументы</param>
        private void MpsFormKeyDown(object sender, KeyEventArgs e)
        {
            if (_secretActivated)
            {
                return;
            }
            if (e.KeyCode == Keys.ShiftKey)
            {
                _shiftKeysCount++;
            }
            else
            {
                _shiftKeysCount = 0;
            }
            if (_shiftKeysCount == 5)
            {
                groupBoxParameters.Visible = true;
                groupBoxParameters.Enabled = true;
                buttonGenerateTupples.Enabled = true;
                generateToolStripMenuItem.Visible = true;
                generateToolStripMenuItem.Enabled = true;
                buttonGenerateTupples.Visible = true;
                _secretActivated = true;
            }
        }
    }
}
```