

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

Брянский государственный технический университет

**Утверждаю**

**Ректор университета**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.Н.Федонин**

**«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2017 г.**

#### ОПЕРАЦИОННЫЕ СИСТЕМЫ

**Диспетчеризация и планирование процессов и потоков**

# Методические указания

# к выполнению лабораторной работы №5

# для студентов очной формы обучения

# специальностей 010503 – «Математическое обеспечение и администрирование информационных систем» и 231000 – «Программная инженерия»

Брянск 2017

УКД 004.43

Операционные системы. Диспетчеризация и планирование процессов и потоков. [Текст] + [Электронный ресурс]: методические указания к выполнению лабораторной работы №5 для студентов очной формы обучения специальностей 010503 – «Математическое обеспечение и администрирование информационных систем» и 231000 – «Программная инженерия». – Брянск: БГТУ, 2017. – 17 с.

Разработал:

К.В.Гулаков

канд. техн. наук, доц.

# Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол № 8 от 2.06.2012)

**Методические указания публикуются в авторской редакции**

# Цель работы

Целью работы является изучение понятия процессов и концепции планирования процессов в операционной системе. Необходимо реализовать имитацию появления процессов в ОС и продемонстрировать применение алгоритма планирования.

# Общие сведения

#### 1. Планирование

На сегодняшний день, ведут речь о двух видах планирования в вычислительных системах: **планировании заданий** и **планировании использования процессора**. **Планирование заданий** появилось в пакетных системах после того, как для хранения сформированных пакетов заданий начали использоваться магнитные диски. Магнитные диски, являясь устройствами прямого доступа, позволяют загружать задания в компьютер в произвольном порядке, а не только в том, в котором они были записаны на диск. Изменяя порядок загрузки заданий в вычислительную систему, можно повысить эффективность ее использования. Процедуру выбора очередного задания для загрузки в машину, т. е. для порождения соответствующего процесса, и назвали планированием заданий. **Планирование использования процессора** впервые возникает в мультипрограммных вычислительных системах, где в состоянии готовности могут одновременно находиться несколько процессов. Именно для процедуры выбора из них одного процесса, который получит процессор в свое распоряжение, т. е. будет переведен в состояние исполнение, мы использовали это словосочетание.

*Планирование заданий* используется в качестве **долгосрочного планирования процессов**. Оно отвечает за порождение новых процессов в системе, определяя ее степень мультипрограммирования, т. е. количество процессов, одновременно находящихся в ней. Если степень мультипрограммирования системы поддерживается постоянной, т. е. среднее количество процессов в компьютере не меняется, то новые процессы могут появляться только после завершения ранее загруженных. Поэтому долгосрочное планирование осуществляется достаточно редко, между появлением новых процессов могут проходить минуты и даже десятки минут. Решение о выборе для запуска того или иного процесса оказывает влияние на функционирование вычислительной системы на протяжении достаточно длительного времени. Отсюда и название этого уровня планирования – *долгосрочное*. В некоторых операционных системах долгосрочное планирование сведено к минимуму или отсутствует вовсе. Так, например, во многих интерактивных системах разделения времени порождение процесса происходит сразу после появления соответствующего запроса. Поддержание разумной степени мультипрограммирования осуществляется за счет ограничения количества пользователей, которые могут работать в системе, и особенностей человеческой психологии. Если между нажатием на клавишу и появлением символа на экране проходит 20–30 секунд, то многие пользователи предпочтут прекратить работу и продолжить ее, когда система будет менее загружена.

*Планирование использования процессора* применяется в качестве **краткосрочного планирования процессов**. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени. Поэтому краткосрочное планирование осуществляется, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т. е. в течение короткого промежутка времени, чем и обусловлено название этого уровня планирования – *краткосрочное*.

*Работа по определению того, в какой момент необходимо прервать выполнение текущего активного потока и какому потоку предоставить возможность выполняться, называется* ***планированием****.*

***Диспетчеризация*** *заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой.*

#### 2. Критерии планирования

Выбор конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, которых мы хотим достичь, используя планирование. К числу таких целей можно отнести:

* *Справедливость*: гарантировать каждому заданию или процессу определенную часть времени использования процессора в компьютерной системе, стараясь не допустить возникновения ситуации, когда процесс одного пользователя постоянно занимает процессор, в то время как процесс другого пользователя фактически не приступал к выполнению.
* *Эффективность*: постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90 процентов.
* *Сокращение полного времени выполнения* (*turnaround time*): обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.
* *Сокращение времени ожидания* (*waiting time*): минимизировать время, которое проводят процессы в состоянии готовность и задания в очереди для загрузки.
* *Сокращение времени отклика* (*response time*): минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей планирования желательно также, чтобы алгоритмы обладали следующими свойствами:

* *Предсказуемость*. Одно и то же задание должно выполняться приблизительно за одно и то же время. Применение алгоритма планирования не должно приводить, к примеру, к извлечению корня квадратного из 4 за сотые доли секунды при одном запуске и за несколько суток при втором запуске.
* *Оптимизация расходов*, связанных с их работой. Если на каждые 100 миллисекунд, выделенных процессу для использования процессора, будет приходиться 200 миллисекунд на определение того, какой именно процесс получит процессор в свое распоряжение, и на переключение контекста, то такой алгоритм, очевидно, использовать не стоит.
* *Равномерная загрузка* ресурсов вычислительной системы, отдавая предпочтение тем процессам, которые будут занимать малоиспользуемые ресурсы.
* *Масштабируемость*, т. е. не сразу теряли работоспособность при увеличении нагрузки. Например, рост количества процессов в системе в два раза не должен приводить к увеличению полного времени выполнения процессов на порядок.

Многие из приведенных выше целей и свойств являются противоречивыми. Улучшая работу алгоритма с точки зрения одного критерия, мы ухудшаем ее с точки зрения другого. Приспосабливая алгоритм под один класс задач, мы тем самым дискриминируем задачи другого класса.

#### 3. Параметры планирования

Необходимость алгоритма планирования зависит от задач, для которых будет использоваться операционная система.

Задачи алгоритмов планирования:

* *Справедливость* - каждому процессу справедливую долю процессорного времени.
* *Контроль* над выполнением принятой политики.
* *Баланс* - поддержка занятости всех частей системы (например, чтобы были заняты процессор и устройства ввода/вывода).
* *Пропускная способность* - количество задач в час.
* *Оборотное время* - минимизация времени на ожидание обслуживания и обработку задач.
* *Использование процесса* - чтобы процессор всегда был занят.
* *Время отклика* - быстрая реакция на запросы.
* *Соразмерность* - выполнение ожиданий пользователя (например, пользователь не готов к долгой загрузке системы).
* *Окончание работы к сроку* - предотвращение потери данных.
* *Предсказуемость* - предотвращение деградации качества в мультимедийных системах (например, потерь качества звука должно быть меньше чем видео).

Для краткосрочного планирования вводятся динамические параметры. Деятельность любого процесса можно представить, как последовательность циклов использования процессора и ожидания завершения операций ввода-вывода. Промежуток времени непрерывного использования процессора носит на английском языке название *CPU burst*, а промежуток времени непрерывного ожидания ввода-вывода - *I/O burst*. Для краткости изложения мы будем использовать термины *CPU burst* и *I/O burst* без перевода. Значения продолжительности последних и очередных *CPU burst* и *I/O burst* являются важными динамическими параметрами процесса.

Для оценки эффективности функционирования алгоритма планирования могут быть применены количественные показатели. Обозначим через t – процессорное время, необходимое процессу для выполнения (будем его называть длительностью процесса).

Обозначим через – *общее время пребывания* процесса в системе. Эту величину – интервал между моментом ввода процесса в систему и моментом получения результатов – также называют иногда временем реакции процесса. Наряду со временем реакции могут быть полезны также и другие показатели.

*Потерянное время* определяет время, в течение которого процесс находился в системе, но не выполнялся.

*Отношение реактивности* показывает долю процессорного времени (времени выполнения) или долю потерянного времени в общем времени реакции.

*Штрафное отношение* показывает, во сколько раз общее время выполнения процесса превышает необходимое процессорное время.

Средние значения величин и могут служить *количественными показателями эффективности*.

#### 4. Алгоритмы планирования процессов

## 4.1 Алгоритм FCFS

**FCFS** (*first come – first serve*; первым пришел – первым обслуживается) – простейший алгоритм, работа, которой понятна из ее названия. Это алгоритм без вытеснения, то есть процесс, выбранный для выполнения на ЦП, не прерывается, пока не завершится (или не перейдет в состояние ожидания по собственной инициативе). FCFS обеспечивает минимум накладных расходов. Среднее потерянное время при применении этого алгоритма не зависит от длительности процесса, но штрафное отношение при равном потерянном времени будет большим для коротких процессов. Поэтому алгоритм FCFS считается лучшим для длинных процессов. Существенным достоинством этого алгоритма наряду с его простотой является то обстоятельство, что FCFS гарантирует отсутствие бесконечного откладывания процессов: любой поступивший в систему процесс будет, в конце концов, выполнен независимо от степени загрузки системы.

## 4.2 Алгоритм RR

**RR** (*round robin* – карусель) – простейший алгоритм с вытеснением. Процесс получает в свое распоряжение ЦП на некоторый квант времени (в простейшем случае размер кванта фиксирован). Если за время процесс не завершился, он вытесняется из ЦП и направляется в конец очереди готовых процессов, где ждет выделения ему следующего кванта, и т.д. Показатели эффективности RR существенно зависят от выбора величины кванта . RR обеспечивает наилучшие показатели, если длительность большинства процессов приближается к размеру кванта, но не превосходит его. Тогда большинство процессов укладываются в один квант и не становятся в очередь повторно. При величине кванта, стремящейся к бесконечности, RR вырождается в *FCFS*. При , стремящемся к 0, накладные расходы на переключение процессов возрастают настолько, что поглощают весь ресурс ЦП. RR обеспечивает наилучшие показатели справедливости: штрафное отношение на большом участке длительностей процессов остается практически постоянным. Только на участке штрафное отношение начинает изменяться и при уменьшении от до возрастает экспоненциально. Потерянное же время существенно растет с увеличением длительности процесса.

## 4.3 Алгоритм SJF

**SJF** (*shortest job first* – самая короткая работа – первой) – невытесняющий алгоритм, в котором наивысший приоритет имеет самый короткий процесс. Для того чтобы применять этот алгоритм, должна быть известна длительность процесса – задаваться пользователем или вычисляться методом экстраполяции. Для коротких процессов SJN обеспечивает лучшие показатели, чем RR, как по потерянному времени, так и по штрафному отношению. SJN обеспечивает максимальную пропускную способность системы – выполнение максимального числа процессов в единицу времени, но показатели для длинных процессов значительно худшие, а при высокой степени загрузки системы активизация длинных процессов может откладываться до бесконечности. Штрафное отношение слабо изменяется на основном интервале значений , но значительно возрастает для самых коротких процессов: такой процесс при поступлении в систему имеет самый высокий приоритет, но вынужден ждать, пока закончится текущий активный процесс.

## 4.4 Алгоритм PSJF

**PSJF** (*preemptive SJN* – *SJN* с вытеснением) – текущий активный процесс прерывается, если его оставшееся время выполнения больше, чем у новоприбывшего процесса. Алгоритм обеспечивает еще большее предпочтение коротким процессам перед длинными. В частности, в ней устраняется то возрастание штрафного отношения для самых коротких процессов, которое имеет место в SJN.

## 4.5 Алгоритм RR SJF

Модификация алгоритма RR с переупорядочиванием процессов в очереди в соответствии с оставшимся временем выполнения.

## 4.6 Приоритетное планирование

Алгоритм *SJF* представляет собой частный случай приоритетного планирования. При приоритетном планировании каждому процессу присваивается определенное числовое значение - приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковыми приоритетами планируются в порядке *FCFS*. Для алгоритма SJF в качестве такого приоритета выступает оценка продолжительности следующего *CPU burst*. Чем меньше значение этой оценки, тем более высокий приоритет имеет процесс.

Принципы назначения приоритетов могут опираться как на внутренние критерии вычислительной системы, так и на внешние по отношению к ней. Внутренние используют различные количественные и качественные характеристики процесса для вычисления его приоритета. Это могут быть, например, определенные ограничения по времени использования процессора, требования к размеру памяти, число открытых файлов и ис-пользуемых устройств ввода-вывода, отношение средних продолжительностей *I/O burst* к *CPU burst* и т. д. Внешние критерии исходят из таких параметров, как важность процесса для достижения каких-либо целей, стоимость оплаченного процессорного времени и других политических факторов.

Планирование с использованием приоритетов может быть как вытесняющим, так и не вытесняющим. При вытесняющем планировании процесс с более высоким приоритетом, появившийся в очереди готовых процессов, вытесняет исполняющийся процесс с более низким приоритетом. В случае не вытесняющего планирования он просто становится в начало очереди готовых процессов. Рассмотрим примеры использования различных режимов приоритетного планирования.

## 4.7 Алгоритм HPRN

**HPRN** (*highest penalty ratio next* – с наибольшим штрафным отношением – следующий) – алгоритм без вытеснения, обеспечивающий наилучшие показатели справедливости. Это достигается за счет динамического переопределения приоритетов. Всякий раз при освобождении ЦП для всех готовых процессов вычисляется текущее штрафное отношение

где – номер процесса; – время, затраченное процессом на ожидание; – длительность процесса, предзаданная или прогнозируемая. Для только что поступившего процесса . ЦП отдается процессу, имеющему наибольшее значение . Для коротких процессов *HPRN* обеспечивает примерно те же показатели справедливости, что и *SJN*, для длинных – более близкие к *FCFS*. На большом диапазоне средних длительностей процессов показатели, обеспечиваемые HPRN, представляют среднее между SJN и FCFS и слабо зависят от длительности. Еще одно достоинство HPRN в том, что во времени ожидания может учитываться (с некоторыми весовыми коэффициентами) и ожидание в других очередях и, таким образом, выполняется более полный учет загрузки системы. Существенным недостатком метода является необходимость перевычисления штрафного отношения для всех процессов при каждом переключении, что плохо согласуется с общей политикой минимизации накладных расходов в дисциплинах без вытеснения.

## 4.8 Алгоритм SRR

**SRR** (*selfish RR* – эгоистичный RR) – метод с вытеснением, дающий дополнительные преимущества выполняемым процессам, что позволяет повысить пропускную способность. Все процессы разделяются на две категории: новые и выбранные. Новыми считаются те процессы, которые не получили еще ни одного кванта времени ЦП, все остальные процессы – выбранные. При поступлении в систему каждому процессу дается некоторый приоритет , одинаковый для всех процессов, который в дальнейшем возрас-тает. В конце каждого кванта времени пересчитываются приоритеты всех процессов, причем приоритеты новых процессов возрастают на величину , а выбранных – на величину . ЦП отдается процессу с наивысшим приоритетом, а при равенстве приоритетов – тому, который раньше поставлен в очередь. Показатели алгоритма существенно зависят от выбранного соотношения между и . При алгоритм вырождается в обыкновенный RR, при – в FCFS. Собственно алгоритм SRR обеспечивается в диапазоне значений .

## 4.9 Лотерейное планирование

Процессам раздаются «лотерейные билеты» на доступ к ресурсам. Планировщик может выбрать любой билет, случайным образом. Чем больше билетов у процесса, тем больше у него приоритет. Распределение квантов времени обычно осуществляют по алгоритму *RR* с приоритетами.

## 4.10 Алгоритм HLRR

**HLRR** (*«half-life round-robin»*). Алгоритм полураспада является модификацией алгоритма RR. С каждым i-м процессом связано некоторое приоритетное число . Чем оно меньше, тем выше приоритет процесса.

Каждый новый процесс получает некоторое исходное значение приоритетного числа , одинаковое для всех процессов. Кроме того, с каждым процессом связан счетчик процессорного времени с исходным значением . Процесс с наименьшим значе-нием получает квант времени (при равенстве приоритетных чисел ЦП отдается процессу, ожидающему дольше). За время кванта интервальный таймер выдает несколько сигналов-прерываний с интервалом . По каждому такому прерыванию счетчик активного (только активного!) процесса увеличивается на 1.

Использование ЦП процессом заканчивается при истечении кванта или при переходе процесса в ожидание. При этом модифицируются счетчики процессорного времени всех (в том числе и неактивных) процессов:

и для всех процессов перевычисляются приоритетные числа:

и модифицируется очередь выполнения процессов.

## 4.11 Многоуровневые очереди (Multilevel Queue)

Для систем, в которых процессы могут быть легко рассортированы на разные группы, был разработан другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процессов, находящихся в состоянии готовность. Этим очередям приписываются фиксированные приоритеты. Например, приоритет очереди системных процессов устанавливается больше, чем приоритет очередей пользовательских процессов. А приоритет очереди процессов, запущенных студентами, - ниже, чем для очереди процессов, запущенных преподавателями. Это значит, что ни один пользовательский процесс не будет выбран для исполнения, пока есть хоть один готовый системный процесс, и ни один студенческий процесс не получит в свое распоряжение процессор, если есть процессы преподавателей, готовые к исполнению. Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов, не требующих взаимодействия с пользователем (фоновых процессов), может использоваться алгоритм *FCFS*, а для интерактивных процессов - алгоритм *RR*. Подобный подход, получивший название многоуровневых очередей, повышает гибкость планирования: для процессов с различными характеристиками применяется наиболее подходящий им алгоритм.

## 4.12 Алгоритм FB

**FB** (*foreground-background* – передний-задний планы) – очередь готовых процессов расщепляется на две подочереди – очередь переднего плана и очередь заднего плана. Очереди обслуживаются по алгоритмам *RR*, но очередь переднего плана имеет абсолютный приоритет: пока в ней есть процессы, очередь заднего плана не обслуживается. Новый процесс направляется в очередь переднего плана. Если процесс использовал установленное число квантов в очереди переднего плана, но не завершился, он переводится в очередь заднего плана.

## 4.12 Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

Дальнейшим развитием алгоритма многоуровневых очередей является добавление к нему механизма обратной связи. Здесь процесс не постоянно приписан к определенной очереди, а может мигрировать из очереди в очередь, в зависимости от своего поведения.

Для простоты рассмотрим ситуацию, когда процессы в состоянии готовность организованы в 4 очереди. Планирование процессов между очередями осуществляется на основе вытесняющего приоритетного механизма. Чем выше располагается очередь, тем выше ее приоритет. Процессы в очереди 1 не могут исполняться, если в очереди 0 есть хотя бы один процесс. Процессы в очереди 2 не будут выбраны для выполнения, пока есть хоть один процесс в очередях 0 и 1. И, наконец, процесс в очереди 3 может получить процессор в свое распоряжение только тогда, когда очереди 0, 1 и 2 пусты. Если при работе процесса появляется другой процесс в какой-либо более приоритетной очереди, исполняющийся процесс вытесняется появившимся. Планирование процессов внутри очередей 0-2 осуществляется с использованием алгоритма *RR*, планирование процессов в очереди 3 основывается на алгоритме *FCFS*.

Родившийся процесс поступает в очередь 0. При выборе на исполнение он получает в свое распоряжение квант времени размером единиц. Если продолжительность его *CPU burst* меньше этого кванта времени, процесс остается в очереди 0. В противном случае, он переходит в очередь 1. Для процессов из очереди 1 квант времени имеет величину . Если процесс не укладывается в это время, он переходит в очередь 2. Если укладывается - остается в очереди 1. В очереди 2 величина кванта времени составляет единицы. Если и этого мало для непрерывной работы процесса, процесс поступает в очередь 3, для которой квантование времени не применяется, и, при отсутствии готовых процессов в других очередях, он может исполняться до окончания своего *CPU burst*. Чем больше значение продолжительности CPU burst, тем в менее приоритетную очередь попадает процесс, но тем на большее процессорное время он может рассчитывать для своего выполнения. Таким образом, через некоторое время все процессы, требующие малого времени работы процессора, окажутся размещенными в высокоприоритетных очередях, а все процессы, требующие большого счета и с низкими запросами к времени отклика, - в низкоприоритетных.

Организация перемещения процессов из очередей с низкими приоритетами в очереди с большими приоритетами позволяет более полно учитывать изменение поведения процессов с течением времени.

# Порядок выполнения лабораторной работы

Вариант задания (алгоритм планирования и параметры) предоставляется преподавателем.

#### Общий план выполнения работы

1. Реализовать имитацию появления процессов в системе случайным образом. Для каждого процесса генерируется время его появление в системе, количество квантов необходимые для работы процесса и, если необходимо, приоритет.

2. Пользователь задает параметры генератора процессов: количество работающих процессов, ограничение на максимальное время выполнения каждого процесса и если необходимо количество допустимых приоритетов.

3. Для полученного графика появления процессов в системе применить алгоритм планирования.

4. Для каждого процесса со временем выполнения t вычислять:

* T – общее время пребывания процесса в системе.
* Потерянное время ;
* Отношение реактивности ;
* Штрафное отношение ;

5. Вычислить средние значения приведенных выше параметров при использовании алгоритма планирования процессов.

6. Для сравнения реализовать также простой алгоритм планирования, указанный в задании. Для простого алгоритма планирования также вычислять средние значения параметров.

7. Осуществить отображение на экране результатов планирования и выполнения процессов виде схемы, графика или таблицы.

# Варианты заданий

|  |  |  |
| --- | --- | --- |
| № | Алгоритм планирования | Простой алгоритм планирования |
| 1. | **SJF с приоритетом** | FCFS |
| 2. | **RR с приоритетом** | PSJF |
| 3. | **RR с приоритетом** | FCFS |
| 4. | **HPRN** | RR |
| 5. | **HPRN** | SJF |
| 6. | **HPRN** | FCFS |
| 7. | **SRR** | FCFS |
| 8. | **SRR** | SJF |
| 9. | **SRR** | RR |
| 10. | **Лотерейное планирование** | FCFS |
| 11. | **Лотерейное планирование** | SJF |
| 12. | **Лотерейное планирование** | RR |
| 13. | **HLRR** | FCFS |
| 14. | **HLRR** | SJF |
| 15. | **HLRR** | RR |
| 16. | **FB** | RR |
| 17. | **FB** | SJF |
| 18. | **MFQ** | FCFS |
| 19. | **MFQ** | SJF |
| 20. | **MFQ** | FB |

# Контрольные вопросы

1. Что такое процесс?
2. Чем планирование заданий отличается от планирования процессов?
3. Какие цели и задачи достигаются планированием?
4. Какие количественные показатели показывают эффективность планирования процессов?
5. В чем отличие планирования от диспетчеризации?

# Список рекомендуемой литературы

1. А. Ю. Молчанов Системное программное обеспечение: /. - СПб.; М.; Нижний Новгород: Питер: Питер принт, 2003. - 395 с.

2. Э. Таненбаум Современные операционные системы. 2-е изд. - СПб.: Питер, 2007. - 1038 с.

3. Х. М. Дейтел, П. Д. Дейтел, Д. Р. Чофнес Операционные системы: [в 2 т.] : пер. с англ./; под ред. С. М. Молявко. - М.: Бином-Пресс, 2006 – 1023 с.