**26.11.2020 ИС2К Операционные системы**

«Основы операционных систем» К.А. Коньков <https://yadi.sk/i/SojomTcdoQ2v_g>

**Тема:** Лекция 3: **Планирование процессов**

(Учебник «Основы операционных систем» К.А.Коньков стр. 26-40)

**Ответить на вопросы: (кратко 3-4 предложения максимум)**

1. Цели планирования

2. Статические параметры и динамические параметры

3. Выбор для исполнения нового процесса из числа находящихся в состоянии готовность в 4 случая (перечислить)

4. Алгоритмы планирования (кратко 1-2 предложения как поняли)

1. First-Come, First-Served (FCFS)
2. Round Robin (RR)
3. Shortest-Job-First (SJF)
4. Гарантированное планирование
5. Приоритетное планирование
6. Многоуровневые очереди (Multilevel Queue)
7. Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

**Ответы присылать** [kuzn117@yandex.ru](mailto:kuzn117@yandex.ru) **можно в ВК -**  **id480169637**

**Пишите тему письма и файл именуйте: Фамилия-ОС-2611**

**Для тех у кого проблемы с Яндекс диском сегодняшнюю тему (**Лекция 3:

Планирование процессов**)**

**выкладываю ниже**

**3. Лекция: Планирование процессов**

В этой лекции рассматриваются вопросы, связанные с различными уровнями планирования процессов в  
операционных системах. Описываются основные цели и критерии планирования, а также параметры, на  
которых оно основывается. Приведены различные алгоритмы планирования.

Я планов наших люблю громадьё...

В. В. Маяковский

Чем тщательнее мы планируем свою деятельность, тем меньше времени остается на ее осуществление.  
Из анналов Госплана

Всякий раз, когда нам приходится иметь дело с ограниченным количеством ресурсов и несколькими их  
потребителями, будь то фонд заработной платы в трудовом коллективе или студенческая вечеринка с не-  
сколькими ящиками пива, мы вынуждены заниматься распределением наличных ресурсов между потре-  
бителями или, другими словами, планированием использования ресурсов. Такое планирование должно  
иметь четко поставленные цели (чего мы хотим добиться за счет распределения ресурсов) и алгоритмы,  
соответствующие целям и опирающиеся на параметры потребителей. Только при правильном выборе  
критериев и алгоритмов можно избежать таких вопросов, как: "Почему я получаю в десять раз меньше,  
чем мой шеф?" или "А где мое пиво?". Настоящая лекция посвящена планированию исполнения процес-  
сов в мультипрограммных вычислительных системах или, иначе говоря, планированию процессов.

**Уровни планирования**

В первой лекции, рассматривая эволюцию компьютерных систем, мы говорили о двух видах планирова-  
ния 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 пря-

Основы операционных систем

**27**

мого 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, мы и назвали планированием заданий. Пла-  
нирование использования процессора впервые возникает в мультипрограммных вычислительных систе-  
мах, где в состоянии готовность могут одновременно находиться несколько процессов. Именно для про-  
цедуры выбора из них одного процесса, который получит процессор в свое распоряжение, т. е. будет пе-  
реведен в состояние исполнение, мы использовали это словосочетание. Теперь, познакомившись с кон-  
цепцией процессов в вычислительных системах, оба вида планирования мы будем рассматривать как  
различные уровни планирования процессов.

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 поддер-  
живается 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 функцио-  
нирование вычислительной системы на протяжении достаточно длительного времени. Отсюда и название  
этого уровня планирования - долгосрочное. В некоторых операционных системах долгосрочное плани-  
рование 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 количе-  
ства 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 пользо-  
ватели предпочтут прекратить работу и продолжить ее, когда система будет менее загружена.

Планирование использования процессора применяется в качестве краткосрочного планирования процес-  
сов. 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 испол-  
нения оказывает влияние на функционирование системы до наступления очередного аналогичного собы-  
тия, т. е. в течение короткого промежутка времени, чем и обусловлено название этого уровня планирова-  
ния - краткосрочное.

В некоторых вычислительных системах бывает выгодно для повышения производительности временно  
удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть  
его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила назва-  
ние swapping, что можно перевести на русский язык как "перекачка", хотя в специальной литературе оно  
употребляется без перевода - свопинг. Когда и какой из процессов нужно перекачать на диск и вернуть  
обратно, решается дополнительным промежуточным уровнем планирования процессов - среднесрочным.

**Критерии планирования и требования к алгоритмам**

Для каждого уровня планирования процессов можно предложить много различных алгоритмов. Выбор  
конкретного алгоритма определяется классом задач, решаемых вычислительной системой, и целями, ко-  
торых мы хотим достичь, используя планирование. К числу таких целей можно отнести следующие:

• Справедливость - гарантировать каждому заданию или процессу определенную часть времени ис-  
пользования 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 начинал выполняться.

• 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%.

• Сокращение полного времени выполнения (turnaround time) - обеспечить минимальное время ме-  
жду стартом процесса или постановкой задания в очередь для загрузки и его завершением.

Основы операционных систем

**28**

• Сокращение времени ожидания (waiting time) - сократить время, которое проводят процессы в со-  
стоянии готовность и задания в очереди для загрузки.

• Сокращение времени отклика (response time) - минимизировать время, которое требуется процес-  
су в интерактивных системах для ответа на запрос пользователя.

Независимо от поставленных целей планирования желательно также, чтобы алгоритмы обладали сле-  
дующими свойствами.

• Были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то  
же время. Применение алгоритма планирования не должно приводить, к примеру, к извлечению  
квадратного корня из 4 за сотые доли секунды при одном запуске и за несколько суток - при вто-  
ром запуске.

• Были связаны с минимальными накладными расходами. Если на каждые 100 миллисекунд, выде-  
ленные процессу для использования процессора, будет приходиться 200 миллисекунд на опреде-  
ление того, какой именно процесс получит процессор в свое распоряжение, и на переключение  
контекста, то такой алгоритм, очевидно, применять не стоит.

• Равномерно загружали ресурсы вычислительной системы, отдавая предпочтение тем процессам,  
которые будут занимать малоиспользуемые ресурсы.

• Обладали масштабируемостью, 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. "В одну телегу впрячь не мож-  
но коня и трепетную лань". Ничего не поделаешь. Такова жизнь.

**Параметры планирования**

Для осуществления поставленных целей разумные алгоритмы планирования должны опираться на какие-  
либо характеристики процессов в системе, заданий в очереди на загрузку, состояния самой вычислитель-  
ной системы, иными словами, на параметры планирования. В этом разделе мы опишем ряд таких пара-  
метров, не претендуя на полноту изложения.

Все параметры планирования можно разбить на две большие группы: статические параметры и динами-  
ческие параметры. Статические параметры не изменяются в ходе функционирования вычислительной  
системы, динамические же, напротив, подвержены постоянным изменениям.

К статическим параметрам вычислительной системы можно отнести предельные значения ее ресурсов  
(размер оперативной памяти, максимальное количество памяти на диске для осуществления свопинга,  
количество подключенных устройств ввода-вывода и т. п.). Динамические параметры системы описыва-  
ют количество свободных ресурсов на данный момент.

К статическим параметрам процессов относятся характеристики, как правило присущие заданиям уже на  
этапе загрузки.

• Каким пользователем запущен процесс или сформировано задание.

• Насколько важной является поставленная задача, т. е. каков приоритет ее выполнения.

• Сколько процессорного времени запрошено пользователем для решения задачи.

• Каково соотношение процессорного времени и времени, необходимого для осуществления опера-  
ций ввода-вывода.

• Какие ресурсы вычислительной системы (оперативная память, устройства ввода-вывода, специ-  
альные библиотеки и системные программы и т. д.) и в каком количестве необходимы заданию.

Алгоритмы долгосрочного планирования используют в своей работе статические и динамические пара-  
метры вычислительной системы и статические параметры процессов (динамические параметры процес-  
сов на этапе загрузки заданий еще не известны). Алгоритмы краткосрочного и среднесрочного планиро-

Основы операционных систем

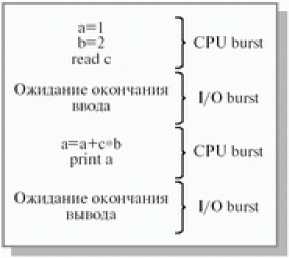
**29**

вания дополнительно учитывают и динамические характеристики процессов. Для среднесрочного плани-  
рования в качестве таких характеристик может использоваться следующая информация:

• сколько времени прошло с момента выгрузки процесса на диск или его загрузки в оперативную  
память;

• сколько оперативной памяти занимает процесс;

• сколько процессорного времени уже предоставлено процессу.



**Рис. 3.1.** Фрагмент деятельности процесса с выделением  
промежутков непрерывного использования процессора и ожидания ввода-вывода

Для краткосрочного планирования нам понадобится ввести еще два динамических параметра. Деятель-  
ность любого процесса можно представить как последовательность циклов использования процессора и  
ожидания завершения операций ввода-вывода. Промежуток времени непрерывного использования про-  
цессора носит название CPU burst, а промежуток времени непрерывного ожидания ввода-вывода - I/O  
burst. На рисунке 3.1. показан фрагмент деятельности некоторого процесса на псевдоязыке программиро-  
вания с выделением указанных промежутков. Для краткости мы будем использовать термины CPU burst  
и I/O burst без перевода. Значения продолжительности последних и очередных CPU burst и I/O burst яв-  
ляются важными динамическими параметрами процесса.

**Вытесняющее и невытесняющее планирование**

Процесс планирования осуществляется частью операционной системы, называемой планировщиком.  
Планировщик может принимать решения о выборе для исполнения нового процесса из числа находящих-  
ся в состоянии готовность в следующих четырех случаях.

1. Когда процесс переводится из состояния исполнение в состояние закончил исполнение.

2. Когда процесс переводится из состояния исполнение в состояние ожидание.

3. Когда процесс переводится из состояния исполнение в состояние готовность (например, после

прерывания от таймера).

4. Когда процесс переводится из состояния ожидание в состояние готовность (завершилась операция  
ввода-вывода или произошло другое событие). Подробно процедура такого перевода рассматри-  
валась в лекции 2 (раздел "Переключение контекста"), где мы показали, почему при этом возника-  
ет возможность смены процесса, находящегося в состоянии исполнение.

В случаях 1 и 2 процесс, находившийся в состоянии исполнение, не может дальше исполняться, и опера-  
ционная система вынуждена осуществлять планирование выбирая новый процесс для выполнения. В  
случаях 3 и 4 планирование может как проводиться, так и не проводиться, планировщик не вынужден  
обязательно принимать решение о выборе процесса для выполнения, процесс, находившийся в состоянии  
исполнение может просто продолжить свою работу. Если в операционной системе планирование осуще-  
ствляется только в вынужденных ситуациях, говорят, что имеет место невытесняющее (nonpreemptive)  
планирование. Если планировщик принимает и вынужденные, и невынужденные решения, говорят о вы-  
тесняющем (preemptive) планировании. Термин "вытесняющее планирование" возник потому, что испол-  
няющийся процесс помимо своей воли может быть вытеснен из состояния исполнение другим процес-  
сом.

Основы операционных систем

**30**

Невытесняющее планирование используется, например, в MS Windows 3.1 и ОС Apple Macintosh. При  
таком режиме планирования процесс занимает столько процессорного времени, сколько ему необходимо.  
При этом переключение процессов возникает только при желании самого исполняющегося процесса пе-  
редать управление (для ожидания завершения операции ввода-вывода или по окончании работы). Этот  
метод планирования относительно просто реализуем и достаточно эффективен, так как позволяет выде-  
лить большую часть процессорного времени для работы самих процессов и до минимума сократить за-  
траты на переключение контекста. Однако при невытесняющем планировании возникает проблема воз-  
можности полного захвата процессора одним процессом, который вследствие каких-либо причин (на-  
пример, из-за ошибки в программе) зацикливается и не может передать управление другому процессу. В  
такой ситуации спасает только перезагрузка всей вычислительной системы.

Вытесняющее планирование обычно используется в системах разделения времени. В этом режиме пла-  
нирования процесс может быть приостановлен в любой момент исполнения. Операционная система ус-  
танавливает специальный таймер для генерации сигнала прерывания по истечении некоторого интервала  
времени - кванта. После прерывания процессор передается в распоряжение следующего процесса. Вре-  
менные прерывания помогают гарантировать приемлемое время отклика процессов для пользователей,  
работающих в диалоговом режиме, и предотвращают "зависание" компьютерной системы из-за зацикли-  
вания какой-либо программы.

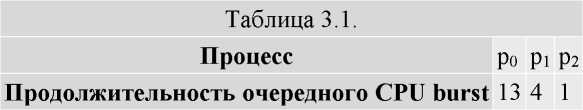
**Алгоритмы планирования**

Существует достаточно большой набор разнообразных алгоритмов планирования, которые предназначе-  
ны для достижения различных целей и эффективны для разных классов задач. Многие из них могут ис-  
пользоваться на нескольких уровнях планирования. В этом разделе мы рассмотрим некоторые наиболее  
употребительные алгоритмы применительно к процессу кратковременного планирования.

**First-Come, First-Served (FCFS)**

Простейшим алгоритмом планирования является алгоритм, который принято обозначать аббревиатурой  
FCFS по первым буквам его английского названия - First-Come, First-Served (первым пришел, первым  
обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь.  
Когда процесс переходит в состояние готовность, он, а точнее, ссылка на его PCB помещается в конец  
этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением от-  
туда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование  
- FIFO1) сокращение от First In, First Out (первым вошел, первым вышел).

Такой алгоритм выбора процесса осуществляет невытесняющее планирование. Процесс, получивший в  
свое распоряжение процессор, занимает его до истечения текущего CPU burst. После этого для выполне-  
ния выбирается новый процесс из начала очереди.



Преимуществом алгоритма FCFS является легкость его реализации, но в то же время он имеет и много  
недостатков. Рассмотрим следующий пример. Пусть в состоянии готовность находятся три процесса ро,  
р1 и р2, для которых известны времена их очередных CPU burst. Эти времена приведены в таблице 3.1. в  
некоторых условных единицах. Для простоты будем полагать, что вся деятельность процессов ограничи-  
вается использованием только одного промежутка CPU burst, что процессы не совершают операций вво-  
да-вывода и что время переключения контекста так мало, что им можно пренебречь.

Если процессы расположены в очереди процессов, готовых к исполнению, в порядке р0, р1, р2, то картина  
их выполнения выглядит так, как показано на рисунке 3.2. Первым для выполнения выбирается процесс  
р0, который получает процессор на все время своего CPU burst, т. е. на 13 единиц времени. После его  
окончания в состояние исполнение переводится процесс р1, он занимает процессор на 4 единицы време-  
ни. И, наконец, возможность работать получает процесс р2. Время ожидания для процесса р0 составляет 0

Основы операционных систем

**31**

единиц времени, для процесса pi - 13 единиц, для процесса р2 - 13 + 4 = 17 единиц. Таким образом, сред-  
нее время ожидания в этом случае - (0 + 13 + 17)/3 = 10 единиц времени. Полное время выполнения для  
процесса р0 составляет 13 единиц времени, для процесса р1 - 13 + 4 = 17 единиц, для процесса р2 - 13 + 4  
+ 1 = 18 единиц. Среднее полное время выполнения оказывается равным (13 + 17 + 18)/3 = 16 единицам  
времени.



**Рис. 3.2.** Выполнение процессов при порядке р0,р1,р2

Если те же самые процессы расположены в порядке р2, р1, р0, то картина их выполнения будет соответст-  
вовать рисунку 3.3. Время ожидания для процесса р0 равняется 5 единицам времени, для процесса р1 - 1  
единице, для процесса р2 - 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы време-  
ни. Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса р0 получа-  
ется равным 18 единицам времени, для процесса р1 - 5 единицам, для процесса р2 - 1 единице. Среднее  
полное время выполнения составляет (18 + 5 + 1)/3 = 8 единиц времени, что почти в 2 раза меньше, чем  
при первой расстановке процессов.



^ ...

**Рис. 3.3.** Выполнение процессов при порядке р2, р1, р0

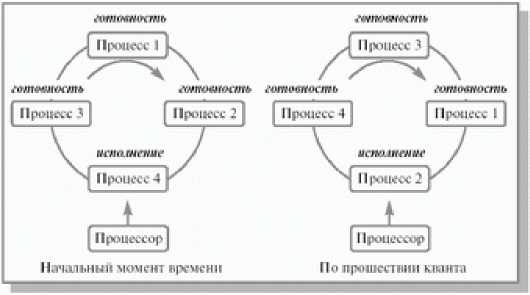
Как мы видим, среднее время ожидания и среднее полное время выполнения для этого алгоритма суще-  
ственно зависят от порядка расположения процессов в очереди. Если у нас есть процесс с длительным  
CPU burst, то короткие процессы, перешедшие в состояние готовность после длительного процесса, бу-  
дут очень долго ждать начала выполнения. Поэтому алгоритм FCFS практически неприменим для систем  
разделения времени - слишком большим получается среднее время отклика в интерактивных процессах.

**Round Robin (RR)**

Модификацией алгоритма FCFS является алгоритм, получивший название Round Robin (Round Robin -  
это вид детской карусели в США) или сокращенно RR. По сути дела, это тот же самый алгоритм, только  
реализованный в режиме вытесняющего планирования. Можно представить себе все множество готовых  
процессов организованным циклически - процессы сидят на карусели. Карусель вращается так, что каж-  
дый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 - 100  
миллисекунд (см. рис. 3.4.). Пока процесс находится рядом с процессором, он получает процессор в свое  
распоряжение и может исполняться.

Основы операционных систем

**32**



**Рис. 3.4.** Процессы на карусели

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся  
в состоянии готовность, в очередь FIFO. Планировщик выбирает для очередного исполнения процесс,  
расположенный в начале очереди, и устанавливает таймер для генерации прерывания по истечении опре-  
деленного кванта времени. При выполнении процесса возможны два варианта.

• Время непрерывного использования процессора, необходимое процессу (остаток текущего CPU  
burst), меньше или равно продолжительности кванта времени. Тогда процесс по своей воле осво-  
бождает процессор до истечения кванта времени, на исполнение поступает новый процесс из на-  
чала очереди, и таймер начинает отсчет кванта заново.

• Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по  
истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов,  
готовых к исполнению, а процессор выделяется для использования процессу, находящемуся в ее  
начале.

Рассмотрим предыдущий пример с порядком процессов p0, pi, p2 и величиной кванта времени равной 4.  
Выполнение этих процессов иллюстрируется таблицей 3.2. Обозначение "И" используется в ней для про-  
цесса, находящегося в состоянии исполнение, обозначение "Г" - для процессов в состоянии готовность,  
пустые ячейки соответствуют завершившимся процессам. Состояния процессов показаны на протяжении  
соответствующей единицы времени, т. е. колонка с номером i соответствует промежутку времени от 0 до  
1.

Таблица 3.2.



Первым для исполнения выбирается процесс p0. Продолжительность его CPU burst больше, чем величина  
кванта времени, и поэтому процесс исполняется до истечения кванта, т. е. в течение 4 единиц времени.  
После этого он помещается в конец очереди готовых к исполнению процессов, которая принимает вид p1,  
p2, p0. Следующим начинает выполняться процесс p1. Время его исполнения совпадает с величиной вы-  
деленного кванта, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоя-  
нии готовность состоит из двух процессов, p2 и p0. Процессор выделяется процессу p2. Он завершается до  
истечения отпущенного ему процессорного времени, и очередные кванты отмеряются процессу p0 -  
единственному не закончившему к этому моменту свою работу. Время ожидания для процесса p0 (коли-  
чество символов "Г" в соответствующей строке) составляет 5 единиц времени, для процесса p1 - 4 едини-  
цы времени, для процесса p2 - 8 единиц времени. Таким образом, среднее время ожидания для этого ал-  
горитма получается равным (5 + 4 + 8)/3 = 5,6(6) единицы времени. Полное время выполнения для про-  
цесса p0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени, для  
процесса p1 - 8 единиц, для процесса p2 - 9 единиц. Среднее полное время выполнения оказывается рав-  
ным (18 + 8 + 9)/3 = 11,6(6) единицы времени.

Основы операционных систем

**33**

Легко увидеть, что среднее время ожидания и среднее полное время выполнения для обратного порядка  
процессов не отличаются от соответствующих времен для алгоритма FCFS и составляют 2 и 6 единиц  
времени соответственно.

На производительность алгоритма RR сильно влияет величина кванта времени. Рассмотрим тот же самый  
пример с порядком процессов p0, pi, p2 для величины кванта времени, равной 1 (см. табл. 3.3.). Время  
ожидания для процесса p0 составит 5 единиц времени, для процесса р1 - тоже 5 единиц, для процесса р2 -  
2 единицы. В этом случае среднее время ожидания получается равным (5 + 5 + 2)/3 = 4 единицам време-  
ни. Среднее полное время исполнения составит (18 + 9 + 3)/3 = 10 единиц времени.

Таблица 3.3.



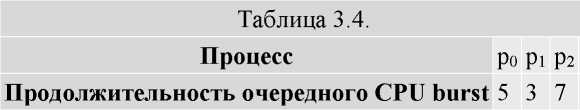
При очень больших величинах кванта времени, когда каждый процесс успевает завершить свой CPU  
burst до возникновения прерывания по времени, алгоритм RR вырождается в алгоритм FCFS. При очень  
малых величинах создается иллюзия того, что каждый из n процессов работает на собственном виртуаль-  
ном процессоре с производительностью ~ 1/n от производительности реального процессора. Правда, это  
справедливо лишь при теоретическом анализе при условии пренебрежения временами переключения  
контекста процессов. В реальных условиях при слишком малой величине кванта времени и, соответст-  
венно, слишком частом переключении контекста накладные расходы на переключение резко снижают  
производительность системы.

**Shortest-Job-First (SJF)**

При рассмотрении алгоритмов FCFS и RR мы видели, насколько существенным для них является поря-  
док расположения процессов в очереди процессов, готовых к исполнению. Если короткие задачи распо-  
ложены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрас-  
тает. Если бы мы знали время следующих CPU burst для процессов, находящихся в состоянии готов-  
ность, то могли бы выбрать для исполнения не процесс из начала очереди, а процесс с минимальной дли-  
тельностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно ис-  
пользовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описан-  
ный алгоритм получил название "кратчайшая работа первой" или Shortest Job First (SJF).

SJF-алгоритм краткосрочного планирования может быть как вытесняющим, так и невытесняющим. При  
невытесняющем SJF-планировании процессор предоставляется избранному процессу на все необходимое  
ему время, независимо от событий, происходящих в вычислительной системе. При вытесняющем SJF-  
планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь  
родившихся или разблокированных) во время работы выбранного процесса. Если CPU burst нового про-  
цесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Рассмотрим пример работы невытесняющего алгоритма SJF. Пусть в состоянии готовность находятся  
четыре процесса, p0, p1, p2 и p3, для которых известны времена их очередных CPU burst. Эти времена при-  
ведены в таблице 3.4. Как и прежде, будем полагать, что вся деятельность процессов ограничивается ис-  
пользованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода  
и что временем переключения контекста можно пренебречь.



Основы операционных систем

**34**

При использовании невытесняющего алгоритма SJF первым для исполнения будет выбран процесс p3,  
имеющий наименьшее значение продолжительности очередного CPU burst. После его завершения для  
исполнения выбирается процесс pi, затем р0 и, наконец, p2. Эта картина отражена в таблице 3.5.

Таблица 3.5.



**Рз И**

Как мы видим, среднее время ожидания для алгоритма SJF составляет (4 + 1 + 9 + 0)/4 = 3,5 единицы  
времени. Легко посчитать, что для алгоритма FCFS при порядке процессов p0, p1, p2, p3 эта величина бу-  
дет равняться (0 + 5 + 8 + 15)/4 = 7 единицам времени, т. е. будет в два раза больше, чем для алгоритма  
SJF. Можно показать, что для заданного набора процессов (если в очереди не появляются новые процес-  
сы) алгоритм SJF является оптимальным с точки зрения минимизации среднего времени ожидания среди  
класса невытесняющих алгоритмов.

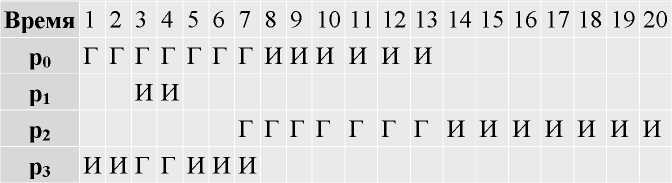
Для рассмотрения примера вытесняющего SJF планирования мы возьмем ряд процессов p0, p1, p2 и p3 с  
различными временами CPU burst и различными моментами их появления в очереди процессов, готовых  
к исполнению (см. табл. 3.6.).

Таблица 3.6.



В начальный момент времени в состоянии готовность находятся только два процесса, p0 и p3. Меньшее  
время очередного CPU burst оказывается у процесса p3, поэтому он и выбирается для исполнения (см.  
таблицу 3.7.). По прошествии 2 единиц времени в систему поступает процесс p1. Время его CPU burst  
меньше, чем остаток CPU burst у процесса p3, который вытесняется из состояния исполнение и перево-  
дится в состояние готовность. По прошествии еще 2 единиц времени процесс p1 завершается, и для ис-  
полнения вновь выбирается процесс p3. В момент времени t = 6 в очереди процессов, готовых к исполне-  
нию, появляется процесс p2, но поскольку ему для работы нужно 7 единиц времени, а процессу p3 оста-  
лось трудиться всего 1 единицу времени, то процесс p3 остается в состоянии исполнение. После его за-  
вершения в момент времени t = 7 в очереди находятся процессы p0 и p2, из которых выбирается процесс  
p0. Наконец, последним получит возможность выполняться процесс p2.

Таблица 3.7.



Основную сложность при реализации алгоритма SJF представляет невозможность точного знания про-  
должительности очередного CPU burst для исполняющихся процессов. В пакетных системах количество  
процессорного времени, необходимое заданию для выполнения, указывает пользователь при формирова-  
нии задания. Мы можем брать эту величину для осуществления долгосрочного SJF-планирования. Если  
пользователь укажет больше времени, чем ему нужно, он будет ждать результата дольше, чем мог бы, так  
как задание будет загружено в систему позже. Если же он укажет меньшее количество времени, задача

Основы операционных систем

**35**

может не досчитаться до конца. Таким образом, в пакетных системах решение задачи оценки времени  
использования процессора перекладывается на плечи пользователя. При краткосрочном планировании  
мы можем делать только прогноз длительности следующего CPU burst, исходя из предыстории работы  
процесса. Пусть т(п) - величина n-го CPU burst, T(n + 1) - предсказываемое значение для п + 1-го CPU  
burst, некоторая величина в диапазоне от 0 до 1.

Определим рекуррентное соотношение

Т (п+1) = “т (n) + (1-“) Т (п)

Т(0) положим произвольной константой. Первое слагаемое учитывает последнее поведение процесса, то-  
гда как второе слагаемое учитывает его предысторию. При “= 0 мы перестаем следить за последним по-  
ведением процесса, фактически полагая

T(n)= T(n+1)=...=T(0)

т. е. оценивая все CPU burst одинаково, исходя из некоторого начального предположения.

Положив “= 1, мы забываем о предыстории процесса. В этом случае мы полагаем, что время очередного  
CPU burst будет совпадать со временем последнего CPU burst:

T(n+1)= т(n)

Обычно выбирают 1Х= 1/2 для равноценного учета последнего поведения и предыстории. Надо отметить,  
что такой выбор “удобен и для быстрой организации вычисления оценки Т(п +1). Для подсчета новой  
оценки нужно взять старую оценку, сложить с измеренным временем CPU burst и полученную сумму  
разделить на 2, например, сдвинув ее на 1 бит вправо. Полученные оценки T(n + 1) применяются как  
продолжительности очередных промежутков времени непрерывного использования процессора для  
краткосрочного SJF-планирования.

**Гарантированное планирование**

При интерактивной работе N пользователей в вычислительной системе можно применить алгоритм пла-  
нирования, который гарантирует, что каждый из пользователей будет иметь в своем распоряжении —1/N  
часть процессорного времени. Пронумеруем всех пользователей от 1 до N. Для каждого пользователя с  
номером i введем две величины: T; - время нахождения пользователя в системе или, другими словами,  
длительность сеанса его общения с машиной и Ti - суммарное процессорное время уже выделенное всем  
его процессам в течение сеанса. Справедливым для пользователя было бы получение Ti/N процессорного  
времени. Если

Ti<<Ti/N

то i-й пользователь несправедливо обделен процессорным временем. Если же

Ti>>Ti/N

то система явно благоволит к пользователю с номером i. Вычислим для процессов каждого пользователя  
значение коэффициента справедливости

TiN/Ti

и будем предоставлять очередной квант времени готовому процессу с наименьшей величиной этого от-  
ношения. Предложенный алгоритм называют алгоритмом гарантированного планирования. К недостат-  
кам этого алгоритма можно отнести невозможность предугадать поведение пользователей. Если некото-  
рый пользователь отправится на пару часов пообедать и поспать, не прерывая сеанса работы, то по воз-  
вращении его процессы будут получать неоправданно много процессорного времени.

Основы операционных систем

**36**

**Приоритетное планирование**

Алгоритмы SJF и гарантированного планирования представляют собой частные случаи приоритетного  
планирования. При приоритетном планировании каждому процессу присваивается определенное число-  
вое значение - приоритет, в соответствии с которым ему выделяется процессор. Процессы с одинаковы-  
ми приоритетами планируются в порядке FCFS. Для алгоритма SJF в качестве такого приоритета высту-  
пает оценка продолжительности следующего CPU burst. Чем меньше значение этой оценки, тем более  
высокий приоритет имеет процесс. Для алгоритма гарантированного планирования приоритетом служит  
вычисленный коэффициент справедливости. Чем он меньше, тем больше у процесса приоритет.

Алгоритмы назначения приоритетов процессов могут опираться как на внутренние параметры, связанные  
с происходящим внутри вычислительной системы, так и на внешние по отношению к ней. К внутренним  
параметрам относятся различные количественные и качественные характеристики процесса такие как:  
ограничения по времени использования процессора, требования к размеру памяти, число открытых фай-  
лов и используемых устройств ввода-вывода, отношение средних продолжительностей I/O burst к CPU  
burst и т. д. Алгоритмы SJF и гарантированного планирования используют внутренние параметры. В ка-  
честве внешних параметров могут выступать важность процесса для достижения каких-либо целей,  
стоимость оплаченного процессорного времени и другие политические факторы. Высокий внешний при-  
оритет может быть присвоен задаче лектора или того, кто заплатил $100 за работу в течение одного часа.

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

Пусть в очередь процессов, находящихся в состоянии готовность, поступают те же процессы, что и в  
примере для вытесняющего алгоритма SJF, только им дополнительно еще присвоены приоритеты (см.  
табл. 3.8.). В вычислительных системах не существует определенного соглашения, какое значение при-  
оритета - 1 или 4 считать более приоритетным. Во избежание путаницы, во всех наших примерах мы бу-  
дем предполагать, что большее значение соответствует меньшему приоритету, т. е. наиболее приоритет-  
ным в нашем примере является процесс p3, а наименее приоритетным - процесс p0.



Как будут вести себя процессы при использовании невытесняющего приоритетного планирования? Пер-  
вым для выполнения в момент времени t = 0 выбирается процесс p3, как обладающий наивысшим при-  
оритетом. После его завершения в момент времени t = 5 в очереди процессов, готовых к исполнению,  
окажутся два процесса р0 и p1. Больший приоритет из них у процесса p1, он и начнет выполняться (см.  
табл. 3.9.). Затем в момент времени t = 8 для исполнения будет избран процесс p2, и лишь потом - про-  
цесс p0.

Таблица 3.9.

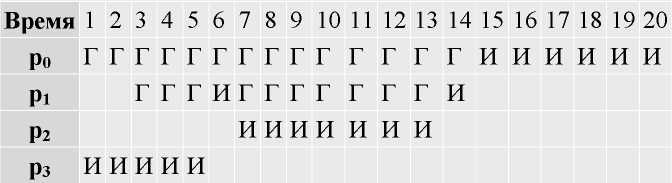


Основы операционных систем

**37**

Иным будет предоставление процессора процессам в случае вытесняющего приоритетного планирования  
(см. табл. 3.10.). Первым, как и в предыдущем случае, начнет исполняться процесс р3, а по его окончании  
- процесс p1. Однако в момент времени t = 6 он будет вытеснен процессом р2 и продолжит свое выполне-  
ние только в момент времени t = 13. Последним, как и раньше, будет исполняться процесс р0.

Таблица 3.10.



В рассмотренном выше примере приоритеты процессов с течением времени не изменялись. Такие при-  
оритеты принято называть статическими. Механизмы статической приоритетности легко реализовать, и  
они сопряжены с относительно небольшими издержками на выбор наиболее приоритетного процесса.  
Однако статические приоритеты не реагируют на изменения ситуации в вычислительной системе, кото-  
рые могут сделать желательной корректировку порядка исполнения процессов. Более гибкими являются  
динамические приоритеты процессов, изменяющие свои значения по ходу исполнения процессов. На-  
чальное значение динамического приоритета, присвоенное процессу, действует в течение лишь коротко-  
го периода времени, после чего ему назначается новое, более подходящее значение. Изменение динами-  
ческого приоритета процесса является единственной операцией над процессами, которую мы до сих пор  
не рассмотрели. Как правило, изменение приоритета процессов проводится согласованно с совершением  
каких-либо других операций: при рождении нового процесса, при разблокировке или блокировании про-  
цесса, по истечении определенного кванта времени или по завершении процесса. Примерами алгоритмов  
с динамическими приоритетами являются алгоритм SJF и алгоритм гарантированного планирования.  
Схемы с динамической приоритетностью гораздо сложнее в реализации и связаны с большими издерж-  
ками по сравнению со статическими схемами. Однако их использование предполагает, что эти издержки  
оправдываются улучшением работы системы.

Главная проблема приоритетного планирования заключается в том, что при ненадлежащем выборе меха-  
низма назначения и изменения приоритетов низкоприоритетные процессы могут не запускаться неопре-  
деленно долгое время. Обычно случается одно из двух. Или они все же дожидаются своей очереди на ис-  
полнение (в девять часов утра в воскресенье, когда все приличные программисты ложатся спать). Или  
вычислительную систему приходится выключать, и они теряются (при остановке IBM 7094 в Массачу-  
сетском технологическом институте в 1973 году были найдены процессы, запущенные в 1967 году и ни  
разу с тех пор не исполнявшиеся). Решение этой проблемы может быть достигнуто с помощью увеличе-  
ния со временем значения приоритета процесса, находящегося в состоянии готовность. Пусть изначально  
процессам присваиваются приоритеты от 128 до 255. Каждый раз по истечении определенного проме-  
жутка времени значения приоритетов готовых процессов уменьшаются на 1. Процессу, побывавшему в  
состоянии исполнение, присваивается первоначальное значение приоритета. Даже такая грубая схема га-  
рантирует, что любому процессу в разумные сроки будет предоставлено право на исполнение.

**Многоуровневые очереди (Multilevel Queue)**

Для систем, в которых процессы могут быть легко рассортированы по разным группам, был разработан  
другой класс алгоритмов планирования. Для каждой группы процессов создается своя очередь процес-  
сов, находящихся в состоянии готовность (см. рис. 3.5). Этим очередям приписываются фиксированные  
приоритеты. Например, приоритет очереди системных процессов устанавливается выше, 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 алго-  
ритмы. Так, например, для больших счетных процессов, не требующих взаимодействия с пользователем  
(фоновых процессов), может использоваться алгоритм FCFS, а для интерактивных процессов - алгоритм

Основы операционных систем

**38**

RR. Подобный подход, получивший название многоуровневых очередей, повышает гибкость планирова-  
ния: для процессов с различными характеристиками применяется наиболее подходящий им алгоритм.



**Рис. 3.5.** Несколько очередей планирования  
**Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)**

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

Для простоты рассмотрим ситуацию, когда процессы в состоянии готовность организованы в 4 очереди,  
как на рисунке 3.6. Планирование процессов между очередями осуществляется на основе вытесняющего  
приоритетного механизма. Чем выше на рисунке располагается очередь, тем выше ее приоритет. Процес-  
сы 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, испол-  
няющийся процесс вытесняется новым. Планирование процессов внутри очередей 0-2 осуществляется с  
использованием алгоритма RR, планирование процессов в очереди 3 основывается на алгоритме FCFS.

ичсрсль 0

Пекле ииершешш

**Приоритет 0**

в всуй\* с клавиатуры

Ккс кнантом времени в

Очередь 1

После завершен *т*

**Приоритет I**

LI! L"K.'i1I501 У I/O

RK с квантом времени 16

**Очередь 2**

После ruBcpiuctfEiH

**Приоритет 2**

других событии

**KR с квантом времени 32**

*у р&жАгмм*

Очередь 3

"-,-Г.|Г■ -||‘- ГI- /Ш,

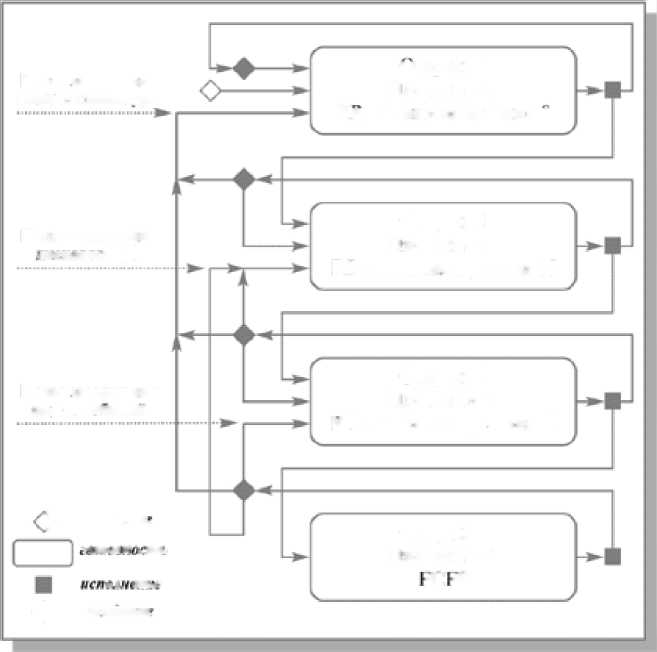
**Приоритет 3**

-С-Ъ

**■■ \* ■си.1'**

♦

*ожтшнт*



Основы операционных систем

**39**

**Рис. 3.6.** Схема миграции процессов в многоуровневых очередях планирования с обратной связью. Вы-  
теснение процессов более приоритетными процессами и завершение процессов на схеме не показано

Родившийся процесс поступает в очередь 0. При выборе на исполнение он получает в свое распоряжение  
квант времени размером 8 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 16. 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 32 единицы.  
Если для непрерывной работы процесса и этого мало, процесс поступает в очередь 3, для которой кван-  
тование времени не применяется и, при отсутствии готовых процессов в других очередях, может испол-  
няться 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.  
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.

Миграция процессов в обратном направлении может осуществляться по различным принципам. Напри-  
мер, после завершения ожидания ввода с клавиатуры процессы из очередей 1, 2 и 3 могут помещаться в  
очередь 0, после завершения дисковых операций ввода-вывода процессы из очередей 2 и 3 могут поме-  
щаться в очередь 1, а после завершения ожидания всех других событий - из очереди 3 в очередь 2. Пере-  
мещение процессов из очередей с низкими приоритетами в очереди с высокими приоритетами позволяет  
более полно учитывать изменение поведения процессов с течением времени.

Многоуровневые очереди с обратной связью представляют собой наиболее общий подход к планирова-  
нию процессов из числа подходов, рассмотренных нами. Они наиболее трудны в реализации, но в то же  
время обладают наибольшей гибкостью. Понятно, что существует много других разновидностей такого  
способа планирования, помимо варианта, приведенного выше. Для полного описания их конкретного во-  
площения необходимо указать:

• Количество очередей для процессов, находящихся в состоянии готовность.

• Алгоритм планирования, действующий между очередями.

• Алгоритмы планирования, действующие внутри очередей.

• Правила помещения родившегося процесса в одну из очередей.

• Правила перевода процессов из одной очереди в другую.

Изменяя какой-либо из перечисленных пунктов, мы можем существенно менять поведение вычислитель-  
ной системы.

На этом мы прекращаем рассмотрение различных алгоритмов планирования процессов, ибо, как было  
сказано: "Нельзя объять необъятное".

**Заключение**

Одним из наиболее ограниченных ресурсов вычислительной системы является процессорное время. Для  
его распределения между многочисленными процессами в системе приходится применять процедуру  
планирования процессов. По степени длительности влияния планирования на поведение вычислительной  
системы различают краткосрочное, среднесрочное и долгосрочное планирование процессов. Конкретные  
алгоритмы планирования процессов зависят от поставленных целей, класса решаемых задач и опираются  
на статические и динамические параметры процессов и компьютерных систем. Различают вытесняющий  
и невытесняющий режимы планирования. При невытесняющем планировании исполняющийся процесс  
уступает процессор другому процессу только по собственному желанию, при вытесняющем планирова-  
нии исполняющийся процесс может быть вытеснен из состояния исполнения помимо своей воли.

Простейшим алгоритмом планирования является невытесняющий алгоритм FCFS, который, однако, мо-  
жет существенно задерживать короткие процессы, не вовремя перешедшие в состояние готовность. В  
системах разделения времени широкое распространение получила вытесняющая версия этого алгоритма  
- RR.

Основы операционных систем

**40**

Среди всех невытесняющих алгоритмов оптимальным с точки зрения среднего времени ожидания про-  
цессов является алгоритм SJF. Существует и вытесняющий вариант этого алгоритма. В интерактивных  
системах часто используется алгоритм гарантированного планирования, обеспечивающий пользователям  
равные части процессорного времени.

Алгоритм SJF и алгоритм гарантированного планирования являются частными случаями планирования с  
использованием приоритетов. В более общих методах приоритетного планирования применяются много-  
уровневые очереди процессов, готовых к исполнению, и многоуровневые очереди с обратной связью. Бу-  
дучи наиболее сложными в реализации, эти способы планирования обеспечивают гибкое поведение вы-  
числительных систем и их адаптивность к решению задач разных классов.