# Алгоритмы планирования процессов

***https://studfile.net/preview/10012472/page:4/***

В ОС принято разделять два вида планирования: планирование заданий и планирование использования процессора. Планирование заданий выступает 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. Планирование использования процессора выступает 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 весьма часто, как правило, не реже одного раза в 100 миллисекунд.

В некоторых вычислительных системах бывает выгодно для повышения их 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 название *swapping*. 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, которых необходимо при этом достичь. 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:

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 корня квадратного из 4 за сотые доли секунды при одном запуске и за несколько суток при втором запуске;

имели минимальные накладные расходы, связанные 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;

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.

Для осуществления поставленных целей алгоритмы планирования должны опираться на какие-либо характеристики процессов в системе, заданий в очереди на загрузку, состояния самой вычислительной системы, иными словами, на параметры планирования.

Все параметры планирования можно разбить на две большие группы: статические параметры и динамические 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 описывают количество свободных ресурсов в текущий момент времени.

## **Вытесняющее и невытесняющее планирование**

Процесс планирования осуществляется частью операционной системы, называемой планировщиком. Планировщик может принимать решения о выборе для исполнения нового процесса, из числа находящихся в состоянии *готовности*, в следующих четырех случаях:

1 когда процесс переводится из состояния *исполнение* в состояние *завершение*;

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).

В случаях 1 и 2 процесс, находившийся в состоянии *исполнение*, не может дальше исполняться, и для выполнения всегда необходимо выбрать новый процесс. В случаях 3 и 4 планирование может не проводиться, процесс, который исполнялся до прерывания, может продолжать свое выполнение после обработки прерывания. Если планирование осуществляется только в случаях 1 и 2, то имеет место *невытесняющее (nonpreemptive) планирование*. В противном случае – *вытесняющее (preemptive) планирование*. Термин “вытесняющее планирование” возник потому, что исполняющийся процесс помимо своей воли может быть вытеснен из состояния *исполнение* другим процессом.

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 передать управление (для ожидания завершения операции ввода-вывода или по окончании работы). Этот метод планирования относительно просто реализуем и достаточно эффективен, так как позволяет использовать большую часть процессорного времени на работу самих процессов и до минимума сократить затраты на переключение контекста. Однако при невытесняющем планировании возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, 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 планирования процесс может быть приостановлен в любой момент своего исполнения. Операционная система устанавливает специальный таймер для генерации сигнала прерывания по истечении некоторого интервала времени – *кванта*. После прерывания процессор передается в распоряжение следующего процесса. Временные прерывания помогают гарантировать приемлемые времена отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают “зависание” компьютерной системы из-за зацикливания какой-либо программы.

## **Алгоритмы планирования**

Существует достаточно большой набор разнообразных алгоритмов планирования, которые предназначены для достижения различных целей и эффективны для разных классов задач. Многие из них могут быть использованы на нескольких уровнях 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*, организованы 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 оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование *FIFO* — 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 своего текущего CPU burst. После этого для выполнения выбирается новый процесс из начала очереди.

Преимуществом алгоритма FCFS является легкость его реализации, в то же время он имеет и много недостатков. Рассмотрим следующий пример. Пусть в состоянии *готовность*находятся три процесса p0, p1 и p2, для которых известны времена их очередных CPU burst. Эти времена приведены в табл. 1 в некоторых условных единицах.

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Процесс | p0 | p1 | p2 |
| Продолжительность очередного CPU burst | 13 | 4 | 1 |

Для простоты полагается, что вся деятельность процессов ограничивается использованием только одного промежутка CPU burst, что процессы не совершают операций ввода-вывода, и что время переключения контекста пренебрежимо мало. Если процессы расположены в очереди процессов готовых к исполнению в порядке p0, p1, p2, то картина их выполнения выглядит так, как показано на рисунке 4.

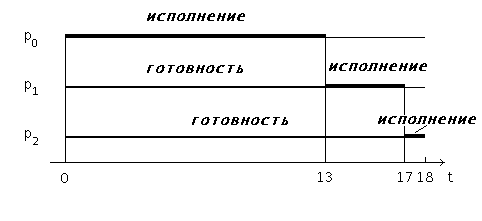


Рис. 4 Выполнение процессов при порядке p0,p1,p2

Первым для выполнения выбирается процесс p0, который получает процессор на все время своего CPU burst, т.е. на 13 единиц времени. После его окончания в состояние *исполнение* переводится процесс p1, занимая процессор на 4 единицы времени. И, наконец, возможность работать получает процесс p2. Время ожидания для процесса p0 составляет 0 единиц времени, для процесса p1 – 13 единиц, для процесса p2—13+4=17 единиц. Таким образом, среднее время ожидания в этом случае — (0+13+17)/3=10 единиц времени. Полное время выполнения для процесса p0 составляет 13 единиц времени, для процесса p1—13+4=17 единиц, для процесса p2—13+4+1=18 единиц. Среднее полное время выполнения оказывается равным (13+17+18)/3=16 единицам времени.

Если те же самые процессы расположены в порядке p2, p1, p0, то картина их выполнения будет соответствовать рисунку 7.

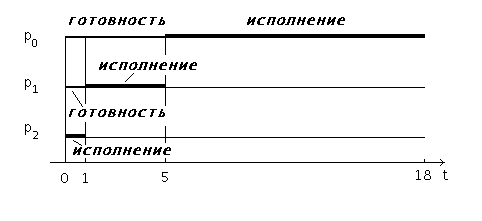


Рис 7. Выполнение процессов при порядке p2,p1,p0

Время ожидания для процесса p0 равняется 5 единицам времени, для процесса p1 — 1 единице, для процесса p2 — 0 единиц. Среднее время ожидания составит (5 + 1 + 0)/3 = 2 единицы времени. Это в 5 раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p0 получается равным 18 единицам времени, для процесса p1 — 5 единицам, для процесса p2 — 1 единице. Среднее полное время выполнения составляет (18 + 5 + 1)/3 = 6 единиц времени, что почти в 2,7 раза меньше чем при первой расстановке процессов.

Как видим, среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно зависят от порядка расположения процессов в очереди. Если есть процесс с длительным CPU burst, то короткие процессы, перешедшие в состояние *готовность* после длительного процесса, будут очень долго ждать начала своего 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 каждый процесс находится около процессора небольшой фиксированный квант времени, обычно 10 - 100 миллисекунд (рис. 5). Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.

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 определенного кванта времени.

При выполнении процесса возможны два варианта:

- время непрерывного использования процессора, требующееся процессу, (остаток текущего CPU burst) меньше или равно продолжительности кванта времени. Тогда процесс по своей воле освобождает процессор до истечения кванта времени, на исполнение выбирается новый процесс из начала очереди и таймер начинает отсчет кванта заново.

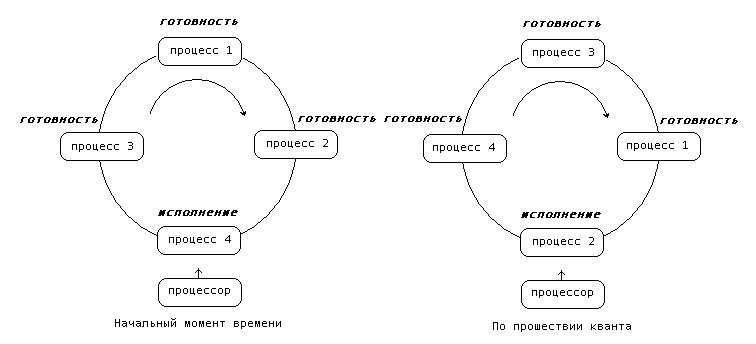


Рис. 5 Процессы на карусели

- 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.

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

Из рассмотрения алгоритмов FCFS и RR видно, насколько существенным для них является порядок расположения процессов в очереди процессов готовых к исполнению. Если короткие задачи расположены в очереди ближе к ее началу, то общая производительность этих алгоритмов значительно возрастает. Если известно время следующих CPU burst для процессов, находящихся в состоянии *готовность*, то можно выбрать для исполнения не процесс из начала очереди, а процесс с минимальной длительностью CPU burst. Если же таких процессов два или больше, то для выбора одного из них можно использовать уже известный нам алгоритм FCFS. Квантование времени при этом не применяется. Описанный 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 требующееся 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 процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.

Основную сложность при реализации алгоритма SJF представляет невозможность точного знания времени очередного CPU burst для исполняющихся процессов. В пакетных системах количество процессорного времени, требующееся заданию для выполнения, указывает пользователь при формировании задания. Величина этого параметра может использоваться для осуществления долгосрочного SJF планирования. Если пользователь укажет больше времени, чем ему нужно, он будет ждать получения результата дольше, чем мог бы, так как задание будет загружено в систему позже. Если же он укажет меньшее количество времени, задача может не досчитаться до конца. Таким образом, в пакетных системах решение задачи оценки времени использования процессора перекладывается на плечи пользователя. При краткосрочном планировании можно делать только прогноз длительности следующего CPU burst, исходя из предыстории работы процесса.

## **Гарантированное планирование**

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

https://studfile.net/html/2706/194/html_bXuKFNp4pZ.gJQP/htmlconvd-eH3ixM_html_f9d99c66fe946bb3.png

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

https://studfile.net/html/2706/194/html_bXuKFNp4pZ.gJQP/htmlconvd-eH3ixM_html_24e1acfb08ebab80.png

то система явно благоволит к пользователю с номером *i*. Вычислим для каждого пользовательского процесса значение коэффициента справедливости

https://studfile.net/html/2706/194/html_bXuKFNp4pZ.gJQP/htmlconvd-eH3ixM_html_cba9843f46dfdab5.png

и будем предоставлять очередной квант времени процессу с наименьшей величиной этого отношения. Предложенный алгоритм называют алгоритмом гарантированного планирования. К недостаткам этого алгоритма можно отнести невозможность предугадать поведение пользователей. Если некоторый пользователь отправится на пару часов пообедать и поспать, не прерывая сеанса работы, то по возвращении его процессы будут получать неоправданно много процессорного времени.

## **Приоритетное планирование**

Алгоритмы SJF и гарантированного планирования представляют собой частные случаи 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. Для алгоритма гарантированного планирования приоритетом служит вычисленный коэффициент справедливости. Чем он меньше, тем больше приоритет у процесса.

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 используемых 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 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.

Главная проблема приоритетного планирования заключается в том, что при ненадлежащем выборе механизма назначения и изменения приоритетов низкоприоритетные процессы могут быть не запущены неопределенно долгое время. Обычно случается одно из двух. Или они все же дожидаются своей очереди на исполнение. Или вычислительную систему приходится выключать, и они теряются (при остановке IBM 7094 в Массачусетском 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 увеличения со временем значения приоритета процесса, находящегося в состоянии ***готовность***.

## **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*** (рис. 6). Этим очередям приписываются фиксированные приоритеты.

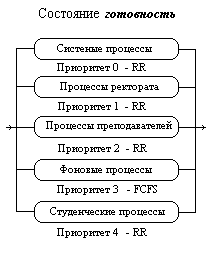


Рис. 6 Несколько очередей планирования

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.

## **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, как на рисунке 7. Планирование процессов между очередями осуществляется на основе вытесняющего приоритетного механизма. Чем выше на рисунке 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.

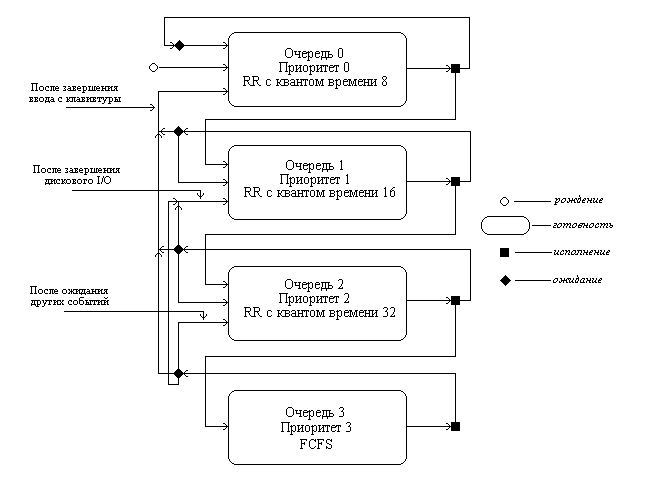


Рис. 7 Схема миграции процессов в многоуровневых очередях

планирования с обратной связью

Многоуровневые очереди с обратной связью представляют собой наиболее общий подход к планированию процессов. Они наиболее трудоемки в реализации, но, в то же время, они обладают наибольшей гибкостью.