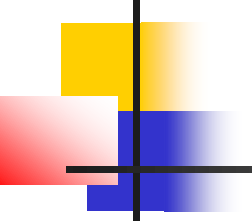


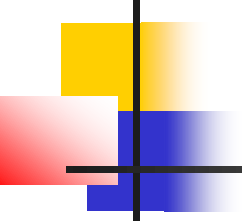
Критерии планирования и требования к алгоритмам

1) 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.



Критерии планирования и требования к алгоритмам

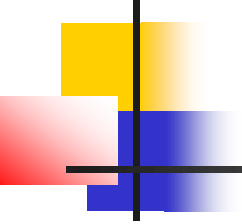
2) Эффективность: постараться занять процессор на все 100% рабочего времени, не позволяя ему простаивать в ожидании процессов готовых к исполнению. В реальных вычислительных системах загрузка процессора колеблется от 40 до 90 процентов.



Критерии планирования и требования к алгоритмам

3) Сокращение полного времени выполнения (turnaround time):

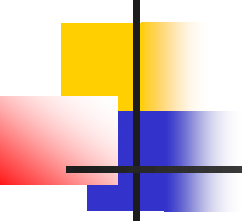
обеспечить минимальное время между стартом процесса или постановкой задания в очередь для загрузки и его завершением.



Критерии планирования и требования к алгоритмам

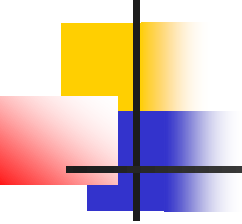
4) Сокращение времени ожидания (waiting time):

минимизировать время, которое проводят процессы в состоянии **ГОТОВНОСТЬ** и задания в очереди для загрузки.



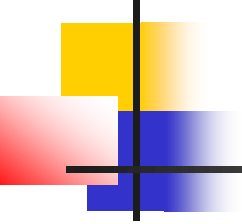
Критерии планирования и требования к алгоритмам

5) Сокращение времени отклика (response time): минимизировать время, которое требуется процессу в интерактивных системах для ответа на запрос пользователя.



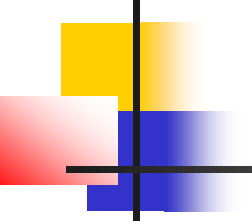
Критерии планирования и требования к алгоритмам

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



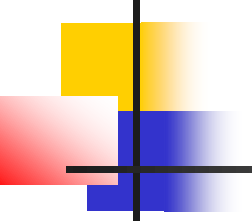
Критерии планирования и требования к алгоритмам

1) Были предсказуемыми. Одно и то же задание должно выполняться приблизительно за одно и то же время.



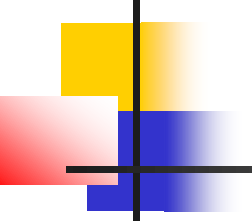
Критерии планирования и требования к алгоритмам

2) 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.



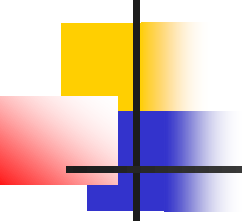
Критерии планирования и требования к алгоритмам

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



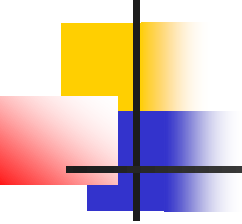
Критерии планирования и требования к алгоритмам

4) 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.**



Параметры планирования

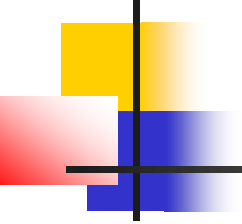
Все параметры планирования можно разбить на две большие группы: статические параметры и динамические параметры. Статические параметры не изменяются в ходе функционирования вычислительной системы, динамические же, напротив, подвержены постоянным изменениям.



Статические параметры процессов

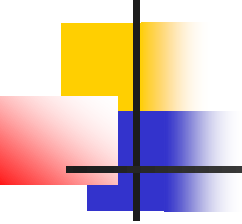
- 1) Каким пользователем запущен процесс или сформировано задание.
- 2) Насколько важной является поставленная задача, т. е. каков приоритет ее выполнения.
- 3) Сколько процессорного времени запрошено пользователем для решения задачи.

Статические параметры процессов



4) Каково соотношение процессорного времени и времени, необходимого для осуществления операций ввода-вывода.

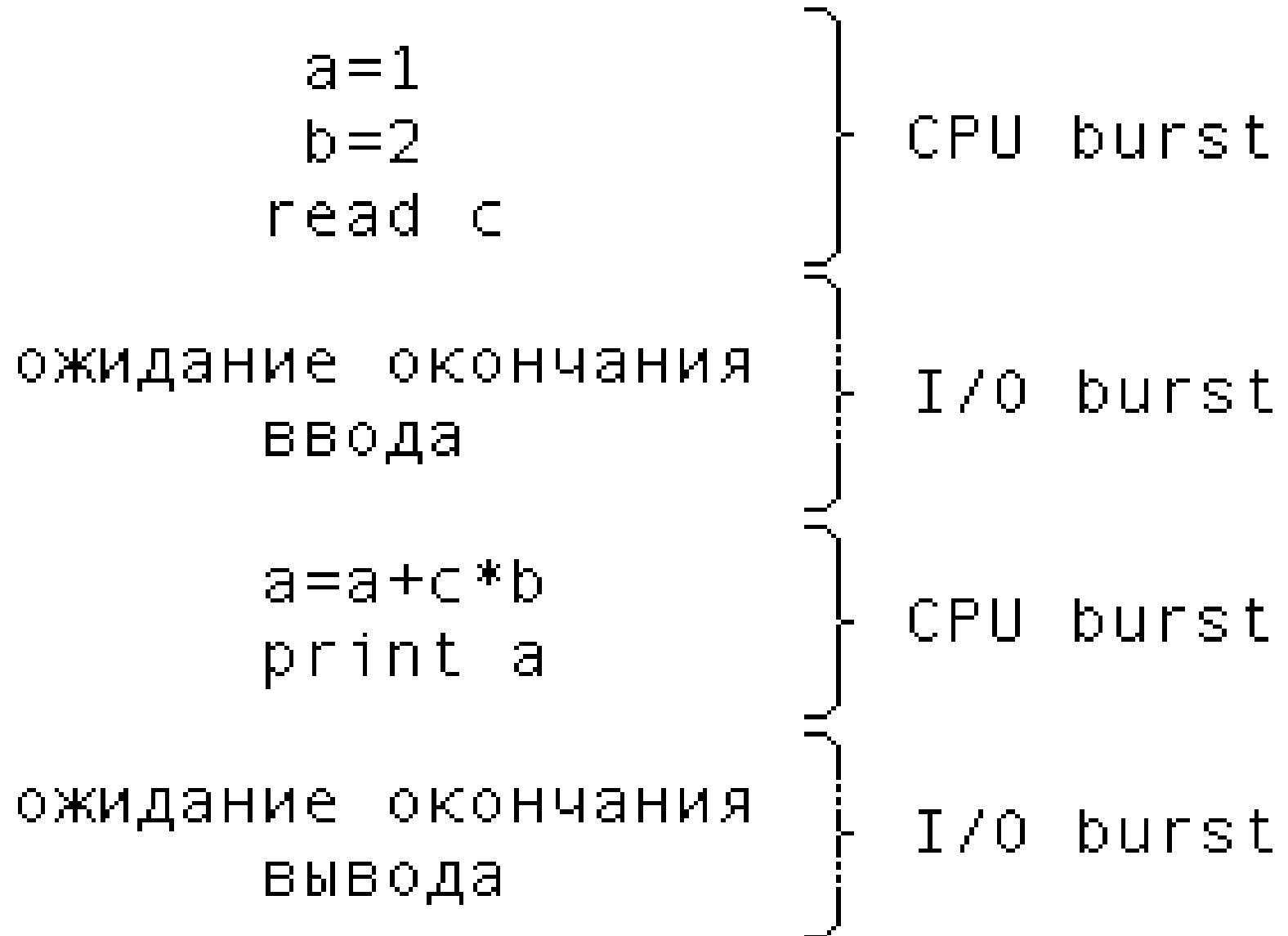
5) Какие ресурсы вычислительной системы (оперативная память, устройства ввода-вывода, специальные библиотеки и системные программы и т. д.) и в каком количестве необходимы заданию.

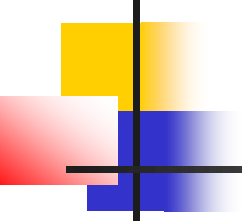


Динамические параметры процессов

- 1) Сколько времени прошло со времени выгрузки процесса на диск или его загрузки в оперативную память.
- 2) Сколько оперативной памяти занимает процесс.
- 3) Сколько процессорного времени было уже предоставлено процессу.

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





Динамические параметры процесса

Деятельность любого процесса
можно представить как
последовательность циклов
использования процессора и
ожидания завершения операций
ввода-вывода.



Динамические параметры процесса

Промежуток времени непрерывного использования процессора носит название *CPU burst*,

Промежуток времени непрерывного ожидания ввода-вывода – *I/O burst*.



Алгоритм 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).



Алгоритм First-Come, First-Served (FCFS)

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).



Алгоритм First-Come, First-Served (FCFS)

Аббревиатура FCFS используется для этого алгоритма планирования вместо стандартной аббревиатуры FIFO для механизмов подобного типа для того, чтобы подчеркнуть, что организация готовых процессов в очередь FIFO возможна и при других алгоритмах планирования



Алгоритм First-Come, First-Served (FCFS)

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.



Алгоритм First-Come, First-Served (FCFS)

Процесс - p_0, p_1, p_2

Продолжительность очередного CPU
burst – 13, 4, 1

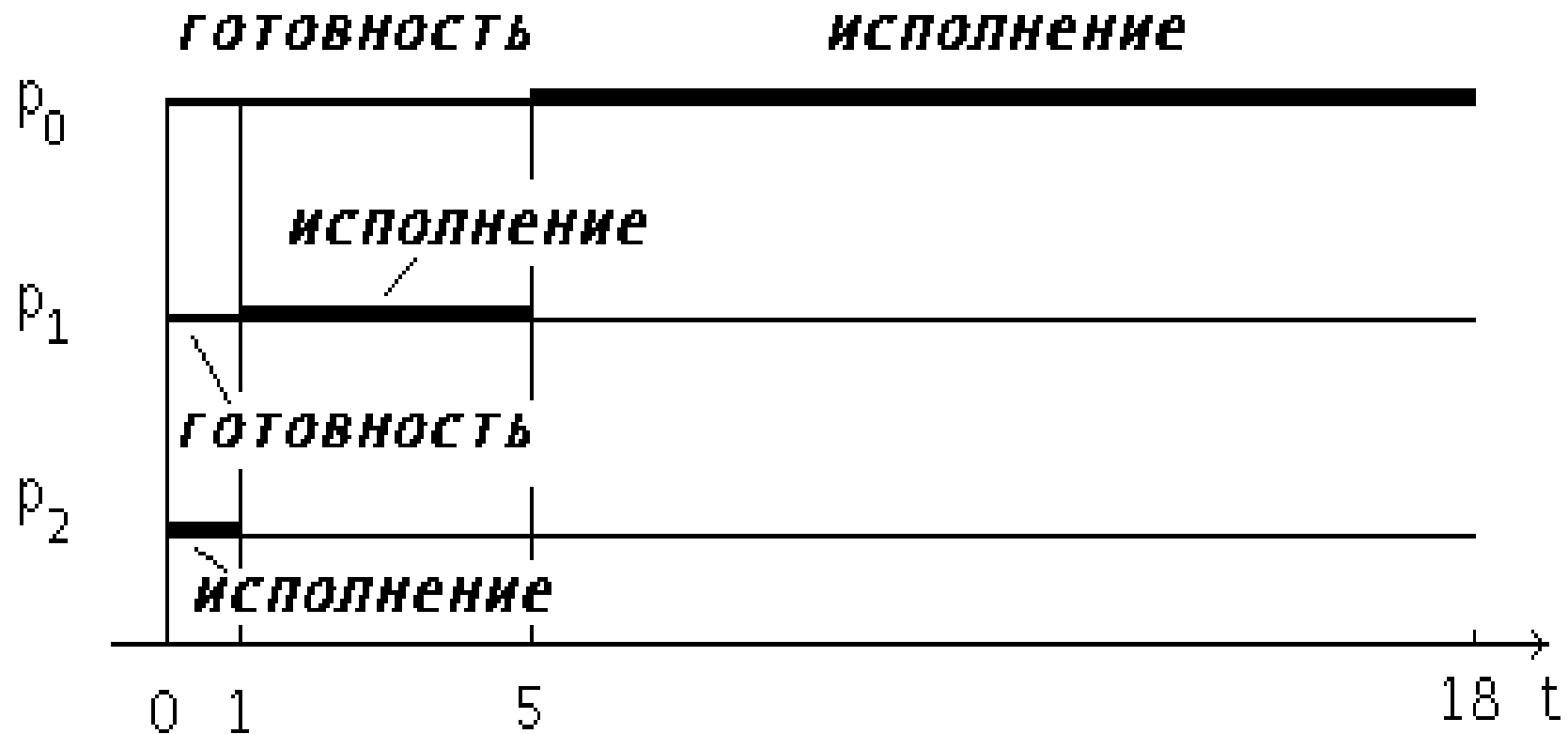
Выполнение процессов при порядке p_0, p_1, p_2



Выполнение процессов при порядке p_0, p_1, p_2

Время ожидания для процесса p_0 составляет 0 единиц времени, для процесса p_1 — 13 единиц, для процесса p_2 — $13 + 4 = 17$ единиц. Таким образом, **среднее время ожидания в этом случае — $(0 + 13 + 17)/3 = 10$ единиц времени.** Полное время выполнения для процесса p_0 составляет 13 единиц времени, для процесса p_1 — $13 + 4 = 17$ единиц, для процесса p_2 — $13 + 4 + 1 = 18$ единиц. **Среднее полное время выполнения оказывается равным $(13 + 17 + 18)/3 = 16$ единицам времени.**

Выполнение процессов при порядке p_2, p_1, p_0



Выполнение процессов при порядке p_2, p_1, p_0

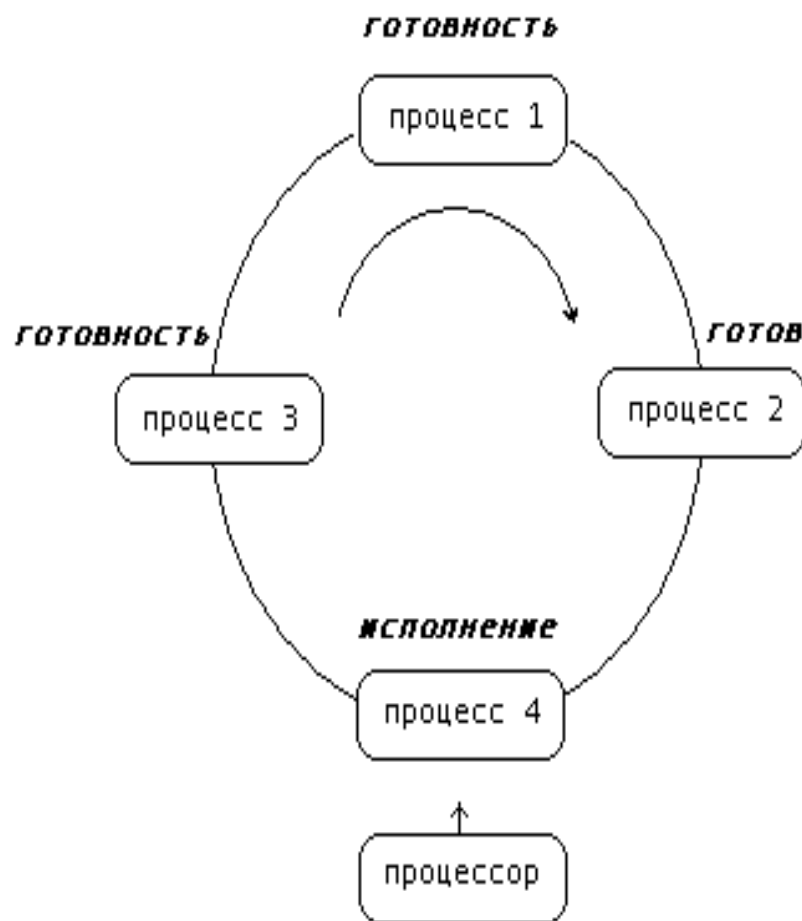
Время ожидания для процесса p_0 равняется 5 единицам времени, для процесса p_1 — 1 единице, для процесса p_2 — 0 единиц. **Среднее время ожидания составит $(5 + 1 + 0)/3 = 2$ единицы времени.** Это в 5 (!) раз меньше, чем в предыдущем случае. Полное время выполнения для процесса p_0 получается равным 18 единицам времени, для процесса p_1 — 5 единицам, для процесса p_2 — 1 единице. **Среднее полное время выполнения составляет $(18 + 5 + 1)/3 = 8$ единиц времени,** что почти в 2,7 раза меньше чем при первой расстановке процессов.



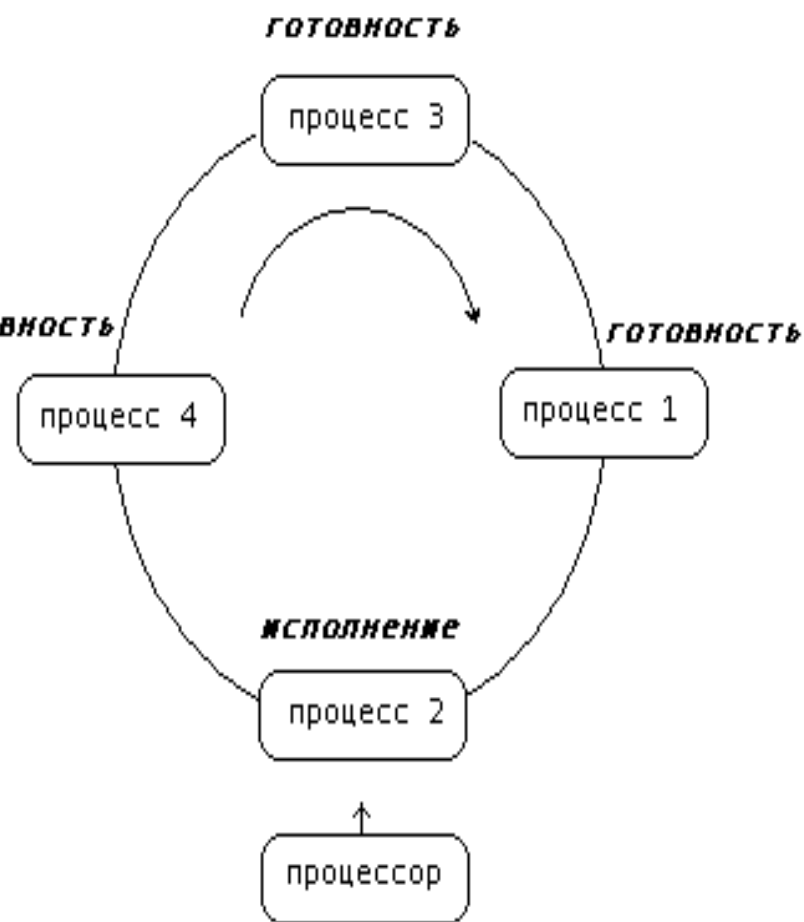
Алгоритм First-Come, First-Served (FCFS) - Выводы

Среднее время ожидания и среднее полное время выполнения для этого алгоритма существенно 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, 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.**

Алгоритм Round Robin (RR)



Начальный момент времени



По прошествии кванта



Алгоритм Round Robin (RR)

Реализуется такой алгоритм так же, как и предыдущий, с помощью организации процессов, находящихся в состоянии ***ГОТОВНОСТЬ***, в очередь FIFO.

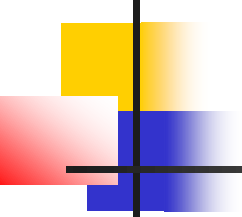
Алгоритм Round Robin (RR)



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:

Алгоритм Round Robin (RR)



1) 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.

Алгоритм Round Robin (RR)

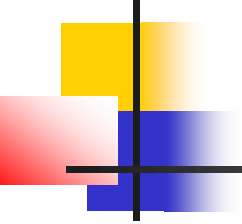


2) Продолжительность остатка текущего CPU burst процесса больше, чем квант времени. Тогда по истечении этого кванта процесс прерывается таймером и помещается в конец очереди процессов ГОТОВЫХ к исполнению, а процессор выделяется для использования процессу, находящемуся в ее начале.

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p0	И	И	И	И	Г	Г	Г	Г	Г	И	И	И	И	И	И	И	И	И
p1	Г	Г	Г	Г	И	И	И	И										
p2	Г	Г	Г	Г	Г	Г	Г	Г	И									

Обозначение **“И”** используется в ней для процесса, 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 1 соответствует промежутку времени от 0 до 1.

Алгоритм Round Robin (RR)



Таким образом, среднее время ожидания для этого алгоритма получается равным $(5 + 4 + 8)/3 = 5,6(6)$ единицы времени.

Полное время выполнения для процесса p_0 (количество непустых столбцов в соответствующей строке) составляет 18 единиц времени, для процесса p_1 — 8 единиц, для процесса p_2 — 9 единиц. Среднее полное время выполнения оказывается равным $(18 + 8 + 9)/3 = 11,6(6)$ единицам времени.

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
p0	И	Г	Г	И	Г	И	Г	И	Г	И	И	И	И	И	И	И	И	И
p1	Г	И	Г	Г	И	Г	И	Г	И									
p2	Г	Г	И															

На производительность алгоритма RR
сильно влияет величина кванта
времени.

На рисунке 1 квант — 1 единица
времени

Алгоритм Round Robin (RR)



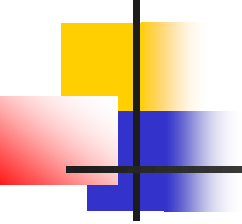
В этом случае среднее время ожидания получается равным $(5 + 5 + 2)/3 = 4$ единицам времени. Среднее полное время исполнения составит $(18 + 9 + 3)/3 = 10$ единиц времени.

Алгоритм Round Robin (RR)



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.

Алгоритм Round Robin (RR)



В реальных условиях при слишком малой величине кванта времени и, соответственно, слишком частом переключении контекста, накладные расходы на переключение резко снижают производительность системы.

Алгоритм Shortest-Job-First (SJF)



SJF алгоритм краткосрочного
планирования может быть как
вытесняющим, так и невытесняющим.



Алгоритм Shortest-Job-First (SJF)

При невытесняющем SJF планировании процессор предоставляется избранному процессу на все требующееся ему время, независимо от событий происходящих в вычислительной системе.



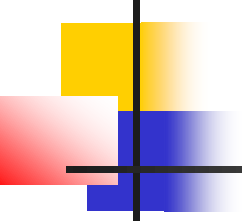
Алгоритм Shortest-Job-First (SJF)

При вытесняющем SJF планировании учитывается появление новых процессов в очереди готовых к исполнению (из числа вновь родившихся или разблокированных) во время работы выбранного процесса.



Алгоритм Shortest-Job-First (SJF)

Если CPU burst нового процесса меньше, чем остаток CPU burst у исполняющегося, то исполняющийся процесс вытесняется новым.



Пример с невытесняющим планированием

Процесс p0, p1, p2, p3

Продолжительность очередного CPU
burst - 5, 3, 7, 1

время	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p0	Г	Г	Г	Г	И	И	И	И	И							
p1	Г	И	И	И												
p2	Г	Г	Г	Г	Г	Г	Г	Г	Г	И	И	И	И	И	И	И
p3	И															

Сначала выполняется p3,
затем p1, затем p0, затем p2

Пример с вытесняющим

Процесс	Время появления в очереди	Продолжительность очередного CPU burst
p0	0	6
p1	2	2
p2	6	7
p3	0	5

время	1	2	3	4	5	6	7	8	9	10
p0	Г	Г	Г	Г	Г	Г	Г	И	И	И
p1			И	И						
p2							Г	Г	Г	Г
p3	И	И	Г	Г	И	И	И			

время	11	12	13	14	15	16	17	18	19	20
p0	И	И	И							
p1										
p2										
p3	Г	Г	Г	И	И	И	И	И	И	И

Алгоритм Shortest-Job-First (SJF)



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. Мы можем брать эту величину для осуществления долгосрочного SJF планирования.

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

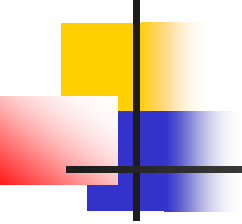
$$t_i \ll \frac{T_i}{N},$$

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

$$t_i \gg \frac{T_i}{N},$$

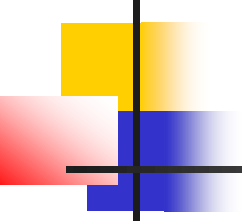
то система явно благоволит к пользователю с номером i . Вычислим для каждого пользовательского процесса значение коэффициента справедливости

$$\frac{t_i N}{T_i}$$



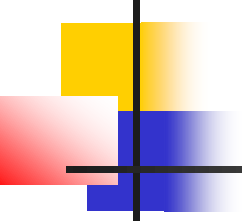
Гарантированное планирование

Если некоторый пользователь отправится на пару часов пообедать, не прерывая сеанса работы, то по возвращении его процессы будут получать неоправданно много процессорного времени.



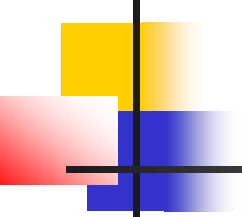
Приоритетное планирование

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.



Приоритетное планирование

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.



Приоритетное планирование

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.

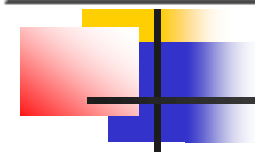
Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
p0	0	6	4
p1	2	2	3
p2	6	7	2
p3	0	5	1

время	1	2	3	4	5	6	7	8	9	10
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г
p1			Г	Г	Г	И	И			
p2							Г	И	И	И
p3	И	И	И	И	И					

время	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	И	И	И	И	И	И
p1										
p2	И	И	И	И						
p3										

Невытесняющее приоритетное планирование

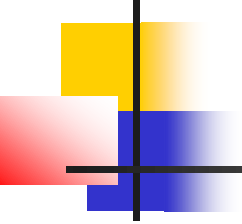
Процесс	Время появления в очереди	Продолжительность очередного CPU burst	Приоритет
p0	0	6	4
p1	2	2	3
p2	6	7	2
p3	0	5	1



время	1	2	3	4	5	6	7	8	9	10
p0	Г	Г	Г	Г	Г	Г	Г	Г	Г	Г
p1			Г	Г	Г	И	Г	Г	Г	Г
p2							И	И	И	И
p3	И	И	И	И	И					

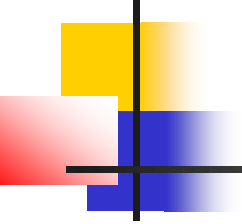
время	11	12	13	14	15	16	17	18	19	20
p0	Г	Г	Г	Г	И	И	И	И	И	И
p1	Г	Г	Г	И						
p2	И	И	И							
p3										

Вытесняющее приоритетное планирование



Приоритетное планирование

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.



Приоритетное планирование

Решение этой проблемы для низкоприоритетных процессов может быть достигнуто с помощью увеличения со временем значения приоритета процесса, находящегося в состоянии **ГОТОВНОСТЬ**.

Пусть изначально процессам присваиваются приоритеты от 128 до 255. Каждый раз, по истечении определенного промежутка времени, значения приоритетов готовых процессов уменьшаются на 1. Процессу, побывавшему в состоянии **ИСПОЛНЕНИЕ**, восстанавливается первоначальное значение приоритета.



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***.



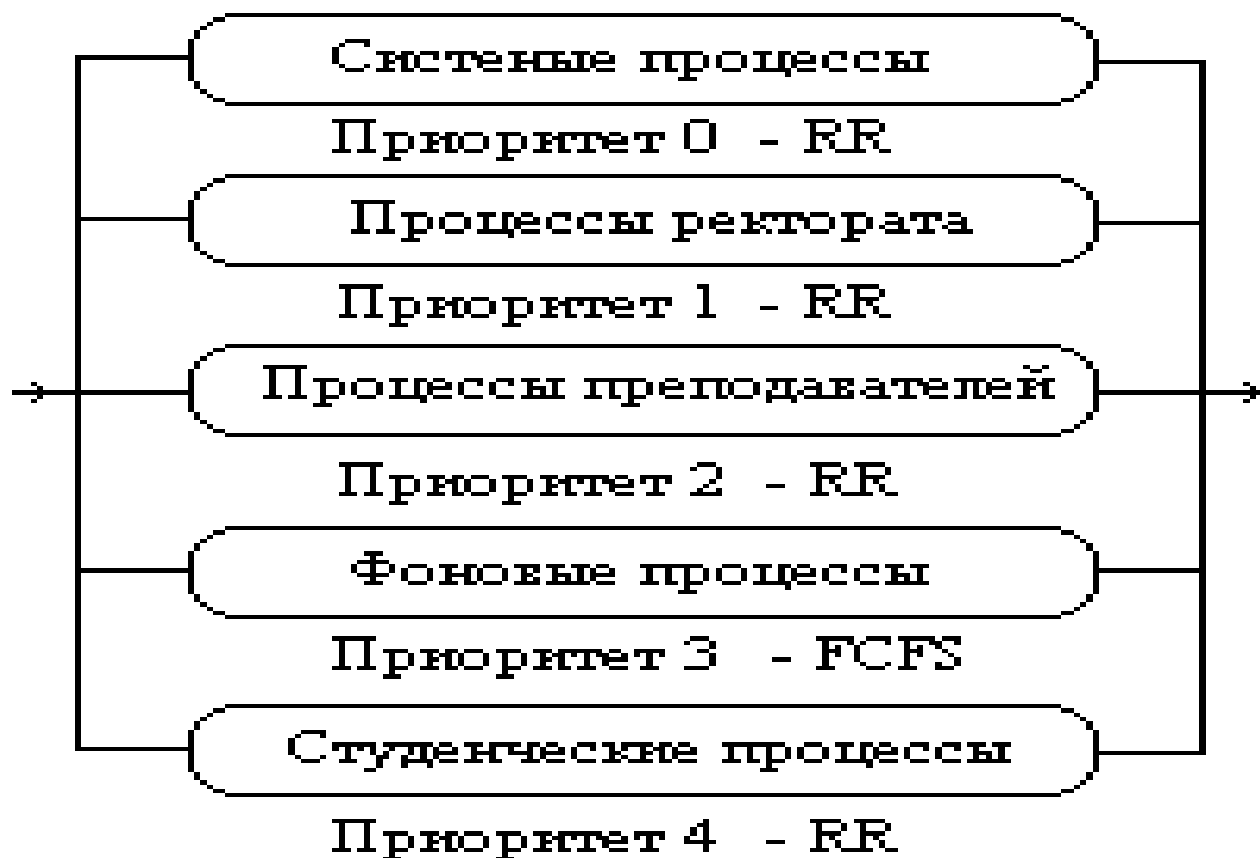
Многоуровневые очереди (Multilevel Queue)

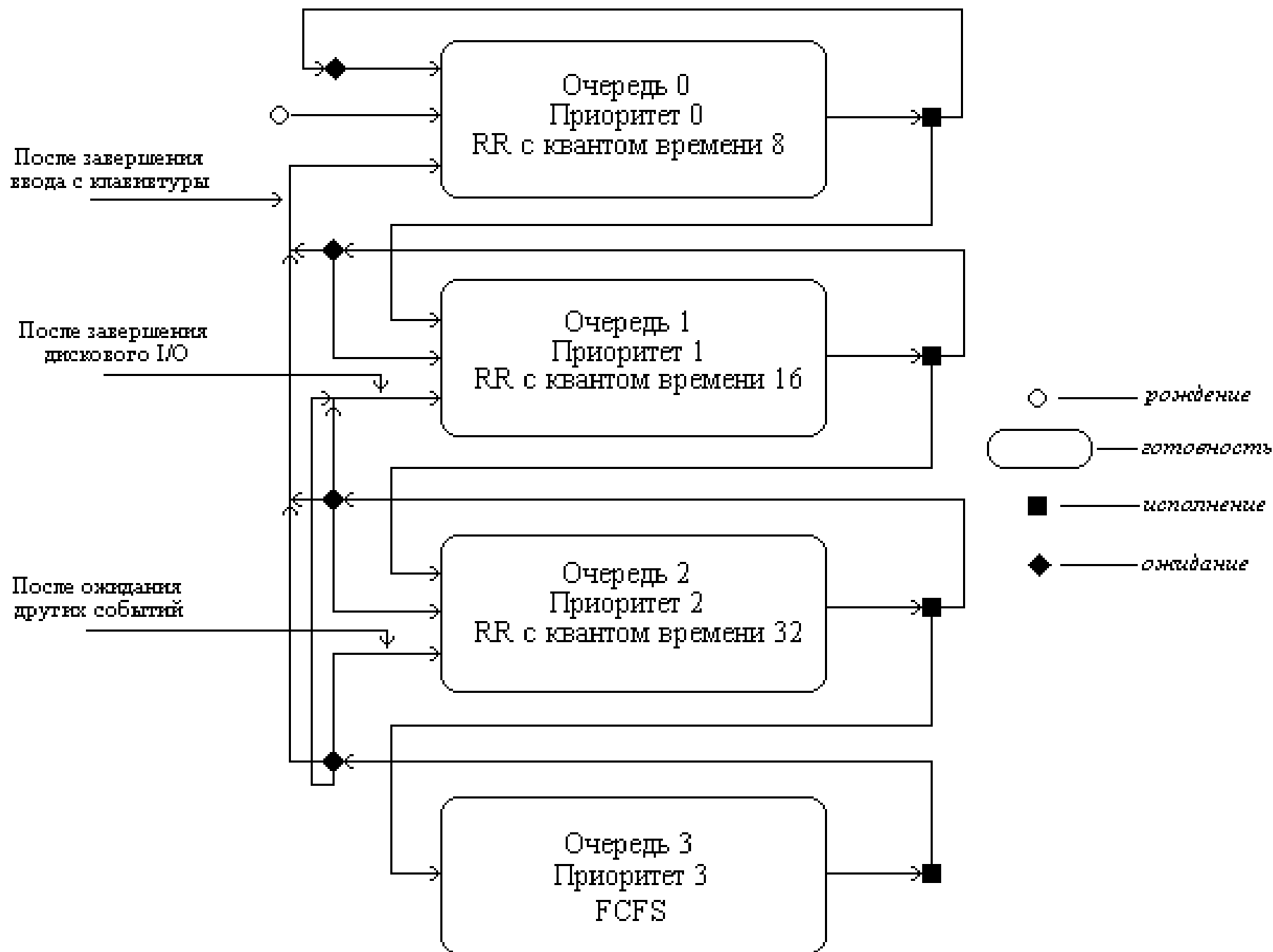
Этим очередям приписываются фиксированные приоритеты.

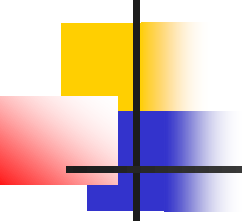
Внутри этих очередей для планирования могут применяться самые разные алгоритмы. Так, например, для больших счетных процессов, не требующих взаимодействия с пользователем (*фоновых* процессов), может использоваться алгоритм FCFS, а для интерактивных процессов – алгоритм RR.

Многоуровневые очереди (Multilevel Queue)

Состояние готовности



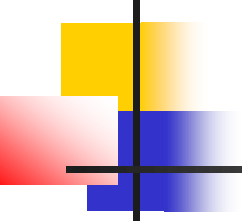




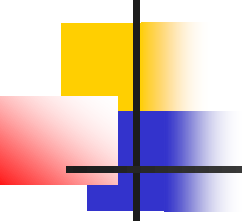
Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

Вытеснение процессов более приоритетными процессами и завершение процессов на схеме не показано.

Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

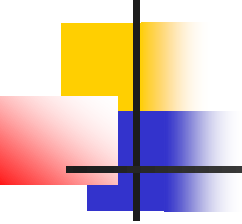


Родившийся процесс поступает в очередь 0. При выборе на исполнение он получает в свое распоряжение квант времени размером 8 единиц. Если продолжительность его CPU burst меньше этого кванта времени, процесс остается в очереди 0. В противном случае, он переходит в очередь 1. Для процессов из очереди 1 квант времени имеет величину 16. Если процесс не укладывается в это время, он переходит в очередь 2. Если укладывается — остается в очереди 1.



Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

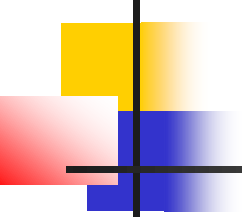
В очереди 2 величина кванта времени составляет 32 единицы. Если и этого мало для непрерывной работы процесса, процесс поступает в очередь 3, для которой квантование времени не применяется, и, при отсутствии готовых процессов в других очередях, он может исполняться до окончания своего CPU burst. Чем больше значение продолжительности CPU burst, тем в менее приоритетную очередь попадает процесс, но тем на большее процессорное время он может рассчитывать для своего выполнения.



Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

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.

Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)



Миграция процессов в обратном направлении может осуществляться по различным принципам. Например, 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 помещаться в очередь 1, а после завершения ожидания всех других событий из очереди 3 в очередь 2. Перемещение процессов из очередей с низкими приоритетами в очереди с большими приоритетами позволяет более полно учитывать изменение поведения процессов с течением времени.



Многоуровневые очереди с обратной связью (Multilevel Feedback Queue)

Для полного описания алгоритма необходимо указать:

- 1) Количество очередей для процессов, находящихся в состоянии **ГОТОВНОСТЬ**.
- 2) Алгоритм планирования, действующий между очередями.
- 3) Алгоритмы планирования, действующие внутри очередей.
- 4) Правила помещения родившегося процесса в одну из очередей.
- 5) Правила перевода процессов из одной очереди в другую.



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.