**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 использоваться на нескольких *уровнях планирования*. В этом разделе мы рассмотрим некоторые наиболее употребительные алгоритмы применительно к процессу кратковременного *планирования*.

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

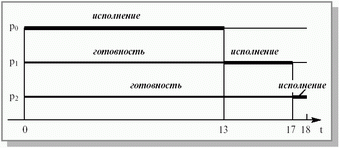
Простейшим алгоритмом *планирования* является алгоритм, который принято обозначать аббревиатурой ***FCFS*** по первым буквам его английского названия – First-Come, First-Served (первым пришел, первым обслужен). Представим себе, что процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда 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 оттуда 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 выполнения выбирается новый процесс из начала очереди.

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

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 p0, p1 и p2, для которых известны времена их очередных *CPU burst* . Эти времена приведены в [таблице 1.](https://intuit.ru/studies/professional_retraining/941/courses/31/lecture/972?page=3#table.3.1) в некоторых условных единицах. Для простоты будем полагать, что вся деятельность процессов ограничивается использованием только одного промежутка *CPU burst* , что процессы не совершают операций ввода-вывода и что время переключения контекста так мало, что им можно пренебречь.

Если процессы расположены в очереди процессов, готовых к исполнению, в порядке p0, p1, p2, то картина их выполнения выглядит так, как показано на рисунке 1. 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, он занимает 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.



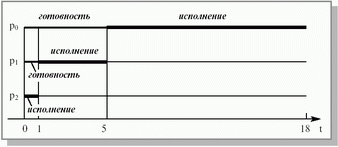
**Рис. 1.** Выполнение процессов при порядке p0, p1,p2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 2. | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| **p0** | И | И | И | И | И | И | И | И | И | И | И | И | И |  |  |  |  |  |
| **p1** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И |  |
| **p2** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И |

***Г – готовность;***

***И – исполнение.***

Если те же самые процессы расположены в порядке p2, p1, p0, то картина их выполнения будет соответствовать рисунку 2. 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 = 8 единиц времени, что почти в 2 раза меньше, чем при первой расстановке процессов.



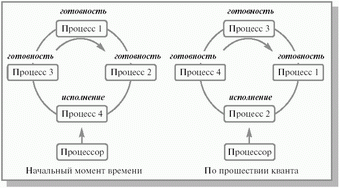
**Рис. 2.** Выполнение процессов при порядке p2, p1, p0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3. | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| **p0** | Г | Г | Г | Г | Г | И | И | И | И | И | И | И | И | И | И | И | И | И |
| **p1** | Г | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p2** | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Как мы видим, 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 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 (см. рис. 3). Пока процесс находится рядом с процессором, он получает процессор в свое распоряжение и может исполняться.



**Рис. 3.** 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 определенного *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 освобождает процессор до истечения *кванта времени*, на исполнение поступает новый процесс из начала 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, p1, p2 и величиной *кванта времени* равной 4. Выполнение этих процессов иллюстрируется таблицей 4. 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 4. | | | | | | | | | | | | | | | | | | | |
| **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 |  |  |  |  |  |  |  |  |  |

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 выделенного *кванта*, поэтому процесс работает до своего завершения. Теперь очередь процессов в состоянии 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 (количество символов "Г" в соответствующей строке) составляет 5 единиц времени, для процесса p1 – 4 единицы времени, для процесса p2 – 8 единиц времени. Таким образом, среднее время ожидания для этого алгоритма получается равным (5 + 4 + 8)/3 = 5,6(6) единицы времени. Полное время выполнения для процесса 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 равным (18 + 8 + 9)/3 = 11,6(6) единицы времени.

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 8 единиц времени соответственно.

На производительность *алгоритма RR* сильно влияет величина *кванта времени*. Рассмотрим тот же самый пример с порядком процессов p0, p1, p2 для величины *кванта времени*, равной 1 (см. табл. 5). Время ожидания для процесса p0 составит 5 единиц времени, для процесса p1 – тоже 5 единиц, для процесса p2 – 2 единицы. 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 5 | | | | | | | | | | | | | | | | | | | |
| **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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

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 виртуальном 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, соответственно, 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 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 бы мы знали время следующих *CPU burst* для процессов, находящихся в состоянии готовность, то могли бы 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 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 процесса 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 приведены в таблице 6. Как и прежде, будем полагать, что вся деятельность процессов ограничивается использованием только одного промежутка *CPU burst*, что процессы не совершают операций ввода-вывода и что временем переключения контекста можно пренебречь.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 6. | | | | |
| **Процесс** | p0 | p1 | p2 | p3 |
| **Продолжительность очередного** ***CPU burst*** | 5 | 3 | 7 | 1 |

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 p1, затем p0 и, наконец, p2. Эта картина отражена в таблице 7.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 7 | | | | | | | | | | | | | | | | |
| **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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

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 будет 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 процессы) *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. 8).

|  |  |  |
| --- | --- | --- |
| Таблица 8 | | |
| **Процесс** | **Время появления в очереди очередного** ***CPU burst*** | **Продолжительность** |
| **p0** | 0 | 6 |
| **p1** | 2 | 2 |
| **p2** | 6 | 7 |
| **p3** | 0 | 5 |

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. табл. 9). 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 переводится в состояние готовность. По прошествии еще 2 единиц времени процесс p1 завершается, и для исполнения вновь выбирается процесс p3. В момент времени t = 6 в очереди процессов, готовых к исполнению, появляется процесс p2, но поскольку ему для работы нужно 7 единиц времени, а процессу p3 осталось трудиться всего 1 единицу времени, то процесс p3 остается в состоянии исполнение. После его завершения 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 9. | | | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| **p0** | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |  |  |  |  |  |  |  |
| **p1** |  |  | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p2** |  |  |  |  |  |  | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И | И |
| **p3** | И | И | Г | Г | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |

Основную сложность при реализации *алгоритма SJF* представляет невозможность точного знания продолжительности очередного *CPU burst* для исполняющихся процессов.

#### 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 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 файлов 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 качестве *внешних параметров* могут выступать важность процесса для достижения каких-либо целей, стоимость оплаченного процессорного времени и другие политические факторы. Высокий внешний *приоритет* 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. 10). В вычислительных системах не существует определенного соглашения, какое значение *приоритета* – 1 или 4 считать более *приоритетным*. Во избежание путаницы, во всех наших примерах мы будем предполагать, что большее значение соответствует меньшему *приоритету*, т. е. наиболее *приоритетным* в нашем примере является процесс p3, а наименее *приоритетным* – процесс p0.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 10. | | | |
| **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 |

Как будут вести себя процессы при использовании *невытесняющего* *приоритетного планирования*? Первым для выполнения в момент времени t = 0 выбирается процесс p3, как обладающий наивысшим *приоритетом*. После его завершения в момент времени t = 5 в очереди процессов, готовых к исполнению, окажутся два процесса p0 и p1. Больший *приоритет* из них у процесса p1, он и начнет выполняться (см. табл. 11). Затем в момент времени t = 8 для исполнения будет избран процесс p2, и лишь потом – процесс p0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 11. | | | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| **p0** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |
| **p1** |  |  | Г | Г | Г | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **p2** |  |  |  |  |  |  | Г | И | И | И | И | И | И | И |  |  |  |  |  |  |
| **p3** | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Иным будет предоставление процессора процессам в случае *вытесняющего* *приоритетного планирования* (см. табл. 12). Первым, как и в предыдущем случае, начнет исполняться процесс p3, а по его окончании – процесс p1. Однако в момент времени t = 6 он будет вытеснен процессом p2 и продолжит свое выполнение только в момент времени t = 13. Последним, как и раньше, будет исполняться процесс p0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 12 | | | | | | | | | | | | | | | | | | | | | |
| **Время** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| **p0** | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | Г | И | И | И | И | И | И |
| **p1** |  |  | Г | Г | Г | И | Г | Г | Г | Г | Г | Г | Г | И |  |  |  |  |  |  |
| **p2** |  |  |  |  |  |  | И | И | И | И | И | И | И |  |  |  |  |  |  |  |
| **p3** | И | И | И | И | И |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

В рассмотренном выше примере *приоритеты* процессов с течением времени не изменялись. Такие *приоритеты* 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, которые могут сделать желательной корректировку порядка исполнения процессов. Более гибкими являются динамические *приоритеты процессов*, изменяющие свои значения по ходу исполнения процессов. Начальное значение динамического *приоритета*, присвоенное процессу, действует в течение лишь короткого периода времени, после чего ему назначается новое, более подходящее значение. Изменение динамического *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 процесса, 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 издержками 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 механизма назначения и изменения *приоритетов* низкоприоритетные процессы могут не запускаться неопределенно долгое время. Обычно случается одно из двух. Или они все же дожидаются своей очереди на исполнение (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 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. 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, присваивается первоначальное значение *приоритета*. Даже такая грубая схема гарантирует, что любому процессу в разумные сроки будет предоставлено право на исполнение.

**Практические задания:**

Для приведенных ниже вариантов выполнить построение таблицы реализации процессов в соответствии с алгоритмом планирования.

1. **Алгоритм FCFS.** Построить таблицу выполнения процессов. Выполнить расчет среднего времени ожидания и полного времени выполнения при исполнении:

p0, p1, p2, p3, p4, p5;

p5, p4, p3, p2, p1, p0;

предложить оптимальный порядок исполнения своего варианта.

1. **Алгоритм RR.** Построить таблицу выполнения процессов. Выполнить расчет среднего времени ожидания и полного времени выполнения при исполнении на двух квантах времени в соответствии с вариантом. Сравнить полученные результаты с заданием 1.
2. **Алгоритм SJF.** Построить таблицу выполнения процессов для *вытесняющего SJF* и *невытесняющего SJF* . Выполнить расчет среднего времени ожидания и полного времени выполнения в соответствии с вариантом. Сравнить с заданием 2.
3. **Алгоритм SJF с приоритетами**. Построить таблицу выполнения процессов для *вытесняющего SJF* и *невытесняющего SJF* . Выполнить расчет среднего времени ожидания и полного времени выполнения в соответствии с вариантом.

***Варианты распределяются в соответствии с последней цифрой в списке группы:***

**1.**

Квант времени 1 = 3

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 3 | 14 | 10 | 5 | 3 | 1 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 0 | 4 |
| **p1** | 2 | 3 |
| **p2** | 6 | 2 |
| **p3** | 0 | 1 |
| **p4** | 0 | 6 |
| **p5** | 3 | 5 |

**2.**

Квант времени 1 = 2

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 8 | 12 | 3 | 1 | 5 | 7 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 2 | 1 |
| **p1** | 2 | 3 |
| **p2** | 4 | 2 |
| **p3** | 0 | 4 |
| **p4** | 0 | 6 |
| **p5** | 3 | 5 |

**3.**

Квант времени 1 = 4

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 4 | 4 | 12 | 10 | 2 | 5 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 7 | 6 |
| **p1** | 2 | 3 |
| **p2** | 6 | 1 |
| **p3** | 0 | 2 |
| **p4** | 0 | 4 |
| **p5** | 3 | 5 |

**4.**

Квант времени 1 = 3

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 8 | 9 | 9 | 1 | 11 | 3 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 0 | 4 |
| **p1** | 1 | 5 |
| **p2** | 7 | 2 |
| **p3** | 10 | 1 |
| **p4** | 0 | 3 |
| **p5** | 13 | 6 |

**5.**

Квант времени 1 = 2

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 10 | 4 | 6 | 7 | 3 | 2 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 0 | 3 |
| **p1** | 2 | 4 |
| **p2** | 6 | 2 |
| **p3** | 4 | 1 |
| **p4** | 7 | 6 |
| **p5** | 6 | 5 |

**6.**

Квант времени 1 = 3

Квант времени 2 = 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 11 | 2 | 6 | 9 | 3 | 1 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 1 | 6 |
| **p1** | 2 | 3 |
| **p2** | 6 | 2 |
| **p3** | 8 | 4 |
| **p4** | 2 | 1 |
| **p5** | 3 | 5 |

**7.**

Квант времени 1 = 2

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 1 | 4 | 9 | 10 | 2 | 2 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 9 | 1 |
| **p1** | 0 | 3 |
| **p2** | 6 | 2 |
| **p3** | 3 | 1 |
| **p4** | 0 | 6 |
| **p5** | 3 | 5 |

**8.**

Квант времени 1 = 4

Квант времени 2 = 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 2 | 8 | 4 | 5 | 12 | 11 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 8 | 1 |
| **p1** | 2 | 3 |
| **p2** | 6 | 4 |
| **p3** | 0 | 2 |
| **p4** | 0 | 6 |
| **p5** | 1 | 5 |

**9.**

Квант времени 1 = 3

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 6 | 1 | 13 | 4 | 5 | 8 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 3 | 1 |
| **p1** | 2 | 2 |
| **p2** | 9 | 3 |
| **p3** | 0 | 4 |
| **p4** | 0 | 6 |
| **p5** | 5 | 5 |

**0.**

Квант времени 1 = 4

Квант времени 2 = 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. | | | | | | |
| **Процесс** | p0 | p1 | p2 | P3 | P4 | P5 | |
| **Продолжительность очередного *CPU burst*** | 12 | 7 | 3 | 8 | 3 | 4 | |

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Время появления в очереди** | ***Приоритет*** |
| **p0** | 7 | 4 |
| **p1** | 12 | 3 |
| **p2** | 4 | 2 |
| **p3** | 11 | 1 |
| **p4** | 4 | 6 |
| **p5** | 3 | 5 |